ELECTRICAL GRID STABILITY STIMULATED DATA USING MACHINE LEARNING ALGORITHMS

A Course Project report submitted

in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

by

THOTA SRIRAM	2103A52070
SYED AJAZ MUKARRAM	2103A52069
NALLELA CHANDU	2103A52062

Under the guidance of

Mr. D. Ramesh

Assistant Professor, Department of CS &AI.

SR UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE &ARTIFICAL INTELLIGENCE
S.R. UNIVERSITY, ANANTHASAGAR, WARANGAL

SR UNIVERSITY

Department of Computer Science and Artifical Intelligence

CERTIFICATE

This is to certify that the Project entitled **"ELECTRICAL GRID STABILITY DATA USING MACHINE LEARNING ALGORITHMS"** is the bonafied work carried out by **THOTA SRIRAM, SYED AJAZ MUKARRAM, NALLELA CHANDU** as a Course Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **ARTIFICAL INTELLIGENCE AND MACHINE LEARNING** during the academic year 2022-2023 under our guidance and Supervision.

.

Mr. D Ramesh	**Dr. M. Sheshikala**
Asst. Professor,	Assoc. Prof. & HOD (CSE),
S R University,	S R University,
Ananthasagar, Warangal.	Ananthasagar, Warangal.

ABSTRACT

The local stability analysis of the 4-node star system (electricity producer is in the center) implementing Decentral Smart Grid Control concept.The main objective and contribution of this paper was/is the application of our knowledge-based data-mining approach (a fuzzy rule-based classification system) characterized by a genetically optimized interpretability-accuracy trade-off (by means of multi-objective evolutionary optimization algorithms) for transparent and accurate prediction of decentral smart grid control (DSGC) stability. In particular, we aim at uncovering the hierarchy of influence of particular input attributes upon the DSGC stability. Moreover, we also analyze the effect of possible "overlapping" of some input attributes over the other ones from the DSGC-stability perspective. The recently published and availableat the 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 was also performed, demonstrating the advantages of our approach in terms of: (i) interpretable and accurate fuzzy rule-based DSGC-stability prediction and (ii) uncovering thehierarchy of DSGC-system's attribute significance.

ACKNOWLEDGEMENT

We express our thanks to Course co-coordinator **Mr.D.Ramesh , Asst. Prof.** for guiding us from the beginning through the end of the Course Project. We express our gratitude to Head of the department CS&AI, **Dr.M.Sheshikala, Associate Professor** for encouragement, support and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved Dean, School of Computer Science and Artificial Intelligence, **Dr C. V. Guru Rao**, for his continuous support and guidance to complete this project in the institute.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

Table of Contents

CHAPTER 1
INTRODUCTION

The stability of electrical grids depends on the balance between electricity generation and electricity demand. In conventional power systems, such a balance is achieved through demand-driven electricity production. Nowadays, however, due to a 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; they are often referred to as prosumers . The volatile and fluctuating nature of renewable energy sources poses a significant challenge asfar as design strategies and control of electric power grids are concerned. In order to balance the supply and demand in such fluctuating power grids, various smart grid approaches have been proposed. A key idea they implement is to regulate the consumers' demand , usually referred to as the demand response strategy .

1.1. PROBLEM STATEMENT

We will make a project for Electrical grid stability simulated data. The problem statement is 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.

1.2. PROPOSED SYSTEM

Goal is to model the data, so as to accurately predict the stability of electric grid. Also, infer the relationship between predictors and the response.The URL of the dataset is https://archive.ics.uci.edu/ml/datasets/Electrical+Grid+Stability+Simulated+Data+.The scope of this project is to apply concepts learntin class on our desire data set. We use R language through out the project.

1.3. PROBLEM DEFINATION OR OBJECTIVE

The objective of this project is to discover patterns in the user data and then make predictions based on given Electrical grid Stability data and intricate patterns to analyze data as well as trends. This project will enable us to formulate machine learning problems corresponding to the dataset . It helps us optimize the machine learning models and report on expected accuracy by applying few methodologies such as Decision Trees and Logistic regression.

CHAPTER 2
LITERATURE SURVEY

Various data-mining and machine-learning models and algorithms have been applied to security, stability prediction/monitoring, management, and controlof electricity grids and microgrids. An approach using an artificial neural network (ANN for short) for generating security boundaries and their visualization to transmission system operators within a power system in

California is presented in [15]. The resulting security boundaries are visualized in the form of the so-called nomograms (the total number of simulations performed is equal to 1792). Extreme learning machines (ELMs for short)—a special class of ANNs—are used in [31] to improve the online learning speed and parameter tuning of the real-time transient-stability assessment model forearlier detection of the risk of blackouts. ANNs are also used to construct new monitoring methods for smart power grids. Such a method and a virtual test evaluating its performance are presented in [32].

A multilayer ANN with four hidden layers trained using a deep reinforcement learning algorithm is applied to a smart grid optimization task in [33]. In turn, a contextual anomaly detection ANN-based approach for cyber-physical security in smart grids is presented in [34]. The simulation experiments show that the contextual anomaly detection performs over 55% better than the point anomaly detection.

CHAPTER 3
ARCHITECTURE

The architecture of this machine learning model is "SUPERVISED LEARNING" and the process involved is data acquisition, data processing, data modelling and execution(parameter tuning and making predictions). The supervised can be further broadened into classification and regression analysis based on output criteria.

Requirement Specification

Hardware Requirements

System	: Pentium4, Intel Core i3, i5, i7 and 2 GHz Minimum
RAM	: 4GB or above
Hard Disk	: 10GB or above
Input	: Keyboard and Mouse
Output	: Monitor or PC

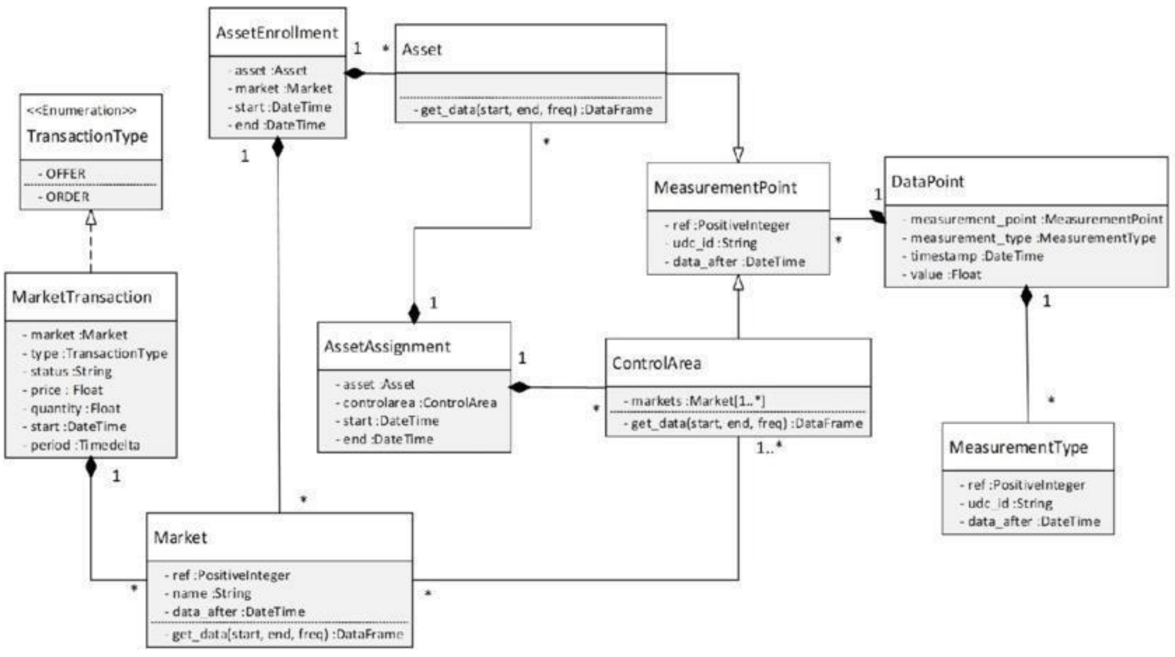
Software Requirements

OS	: Windows 8 or Higher Versions
Platform	: Google colab
Program Language	: Python

CHAPTER 4
DATA PRE-PROCESSING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model. It is the first and crucial stepwhile creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data and while doing any operation with data. It is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task. In this particular section we re-label and convert some categorical features into numericvalues. This is crucial for training machine learning models since machine learning models accepts the numeric values.

4.1. UML DIAGRAM



4.2. DATASET DESCRIPTION:

For this project, we have obtained my dataset from Kaggle. This dataset contains 10000 rows of data and 14columns (features) that we could focus onto build our prediction model i.e., we use 14 attributes to predict the Electrical Grid Stability.

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)

4.3. DATASET CLEANING:

Data Cleaning means the process of identifying the incorrect, incomplete, inaccurate, irrelevant or missing part of the data and then modifying, replacing or deleting them according to thenecessity. Data cleaning is considered a foundational element of the basic data science. Data is the most valuable thing for Analytics and Machine learning. In computing or Business data is needed everywhere.

When it comes to the real world data, it is not improbable that data may contain incomplete, inconsistent or missing values. If the data is corrupted then it may hinder the process or provide inaccurate results.

We have a total of 1000 rows and 14 columns (attributes) in the dataset. We also do feature engineering by the following steps to clean the data.

- Remove extra unnecessary details from grid Columns
- Convert stab string into binary values
- Remove dummy data from the dataset
- Remove double or repeated values from the dataset.

4.4. DATA VISUALISATION

```
import numpy as np
import pandas as pd
                                                                 In [5]:
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
force_remount=True).
```

```
                                                                 In [6]:
my_data = pd.read_csv("Data_for_UCI_named.csv", delimiter=",")my_data[0:13]
```

```
                                                                          ut[6]:
        tau1     tau2     tau3     tau4     p1      p2      p3      p4      g1      g2      g3      g4
                                                                                          stab    stabf
0       2.959060          3.079885          8.381025        9.780754        3.763085          -0.782604
          -1.257395         -1.723086         0.650456        0.859578          0.887445        0.958034
                                                                            0.055347        unstable
```

```
1       9.304097      4.902524      3.047541      1.369357      5.067812      -1.940058
         -1.872742     -1.255012      0.413441      0.862414      0.562139      0.781760
                                                                   -0.005957       stable
2       8.971707      8.848428      3.046479      1.214518      3.405158      -1.207456
         -1.277210     -0.920492      0.163041      0.766689      0.839444      0.109853
                                                                 0.003471      unstable
3       0.716415      7.669600      4.486641      2.340563      3.963791      -1.027473
         -1.938944     -0.997374      0.446209      0.976744      0.929381      0.362718
                                                                 0.028871      unstable
4       3.134112      7.608772      4.943759      9.857573      3.525811      -1.125531
         -1.845975     -0.554305      0.797110      0.455450      0.656947      0.820923
                                                                 0.049860      unstable
5       6.999209      9.109247      3.784066      4.267788      4.429669      -1.857139
         -0.670397     -1.902133      0.261793      0.077930      0.542884      0.469931
                                                                   -0.017385       stable
6       6.710166      3.765204      6.929314      8.818562      2.397419      -0.614590
         -1.208826     -0.574004      0.177890      0.397977      0.402046      0.376630
                                                                 0.005954      unstable
7       6.953512      1.379125      5.719400      7.870307      3.224495      -0.748998
         -1.186517     -1.288980      0.371385      0.633204      0.732741      0.380544
                                                                 0.016634      unstable
8       4.689852      4.007747      1.478573      3.733787      4.041300      -1.410344
         -1.238204     -1.392751      0.269708      0.250364      0.164941      0.482439
                                                                   -0.038677       stable
9       9.841496      1.413822      9.769856      7.641616      4.727595      -1.991363
         -0.857637     -1.878594      0.376356      0.544415      0.792039      0.116263
                                                                 0.012383      unstable
10      5.930110      6.730873      6.245138      0.533288      2.327092      -0.702501
         -1.116920     -0.507671      0.239816      0.563110      0.164461      0.753701
                                                                   -0.028411       stable
11      5.381299      8.014521      8.095174      6.769248      5.507551      -1.972714
         -1.849333     -1.685505      0.359974      0.173569      0.349144      0.628860
                                                                 0.028130      unstable
12      1.616787      2.939228      0.819791      4.191804      3.752282      -1.484885
         -1.280581     -0.986816      0.899698      0.866546      0.303921      0.077610
                                                                   -0.048617       stable
```

```
my_data=my_data.fillna(0)
my_data.head()
```

```
                                                                                  Out[7]:

        tau1     tau2     tau3     tau4     p1       p2       p3       p4       g1       g2       g3       g4
                                                                                              stab     stabf
0       2.959060          3.079885          8.381025          9.780754          3.763085          -0.782604
         -1.257395         -1.723086          0.650456          0.859578          0.887445          0.958034
                                                                                 0.055347          unstable
1       9.304097          4.902524          3.047541          1.369357          5.067812          -1.940058
         -1.872742         -1.255012          0.413441          0.862414          0.562139          0.781760
                                                                                   -0.005957          stable
```

```
2      8.971707     8.848428     3.046479     1.214518     3.405158     -1.207456
         -1.277210    -0.920492     0.163041     0.766689     0.839444     0.109853
                                                               0.003471     unstable
3      0.716415     7.669600     4.486641     2.340563     3.963791     -1.027473
         -1.938944    -0.997374     0.446209     0.976744     0.929381     0.362718
                                                               0.028871     unstable
4      3.134112     7.608772     4.943759     9.857573     3.525811     -1.125531
         -1.845975    -0.554305     0.797110     0.455450     0.656947     0.820923
                                                               0.049860     unstable
```

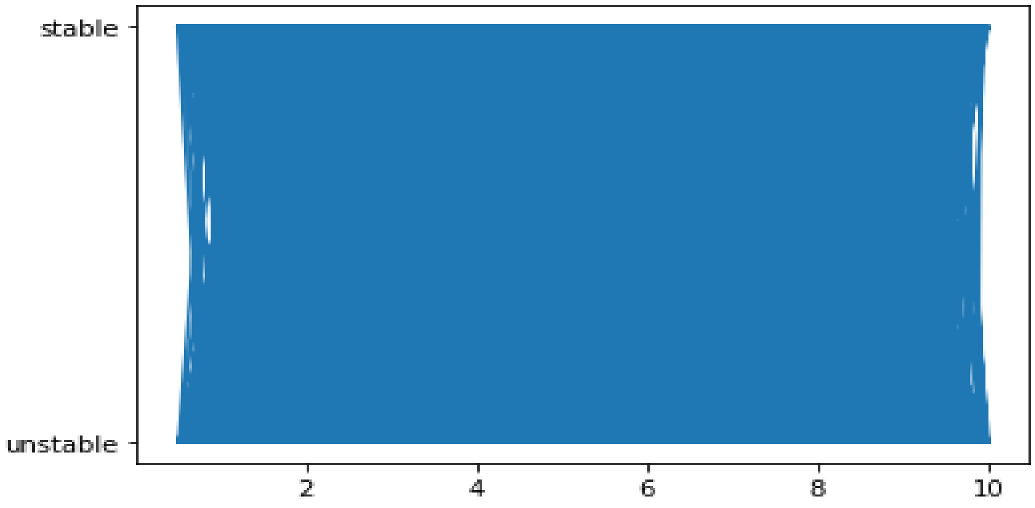
In [8]:

In [9]:

```
y=my_data['stabf']y=y.fillna(0)

x1=my_data['tau1']x1=x1.fillna(0)
from matplotlib import pyplot as ptpt.plot(x1,y)
```

Out[9]:

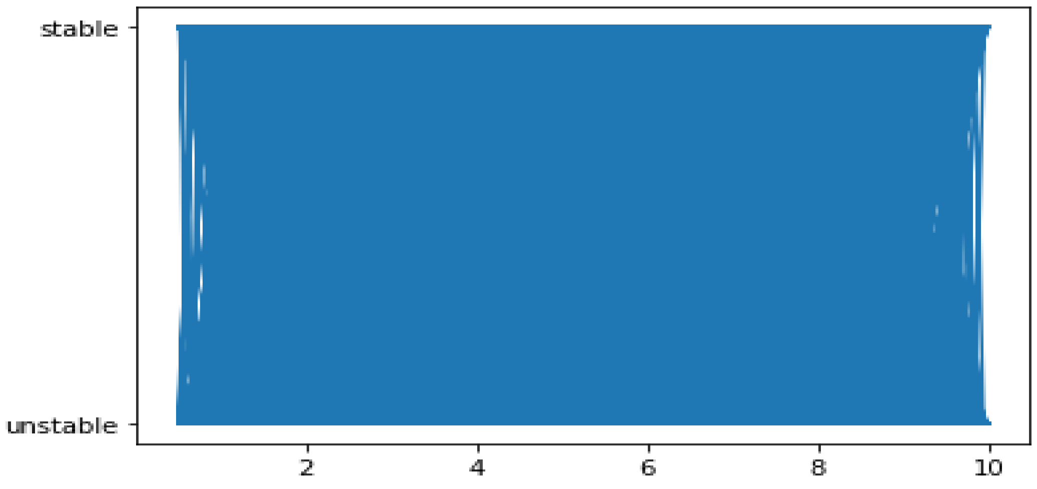
```
[<matplotlib.lines.Line2D  at  0x7f997e0c9d90>]
```



```
x2=my_data['tau2']x2=x2.fillna(0)
from matplotlib import pyplot as ptpt.plot(x2,y)
```

Out[10]:

```
[<matplotlib.lines.Line2D  at  0x7f997dbc5610>]
```

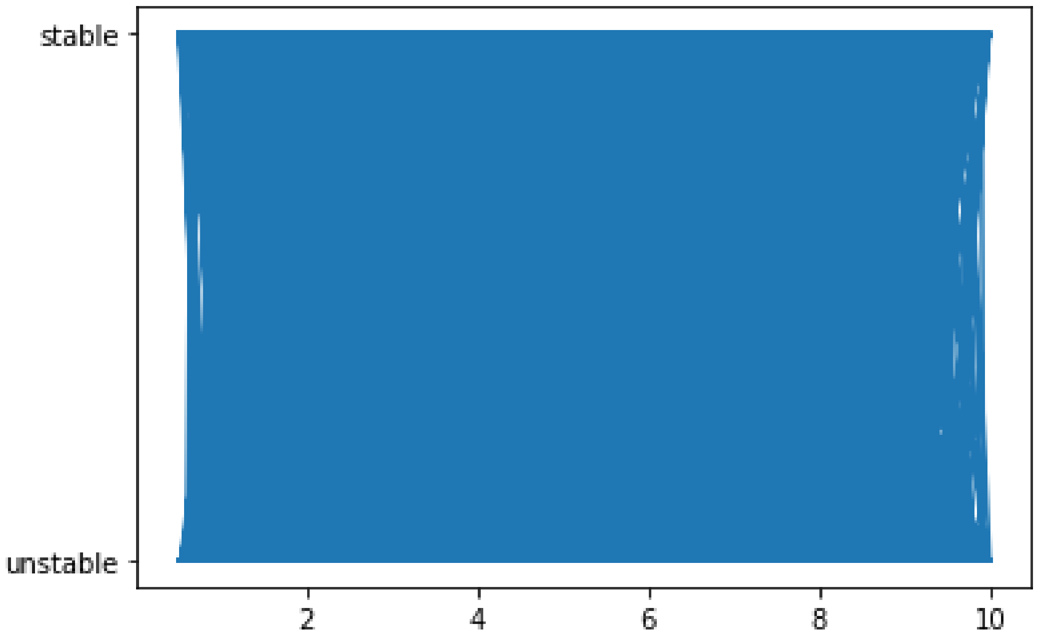


In [11]:

```
x3=my_data['tau3']x3=x3.fillna(0)
from matplotlib import pyplot as ptpt.plot(x3,y)
```

Out[11]:

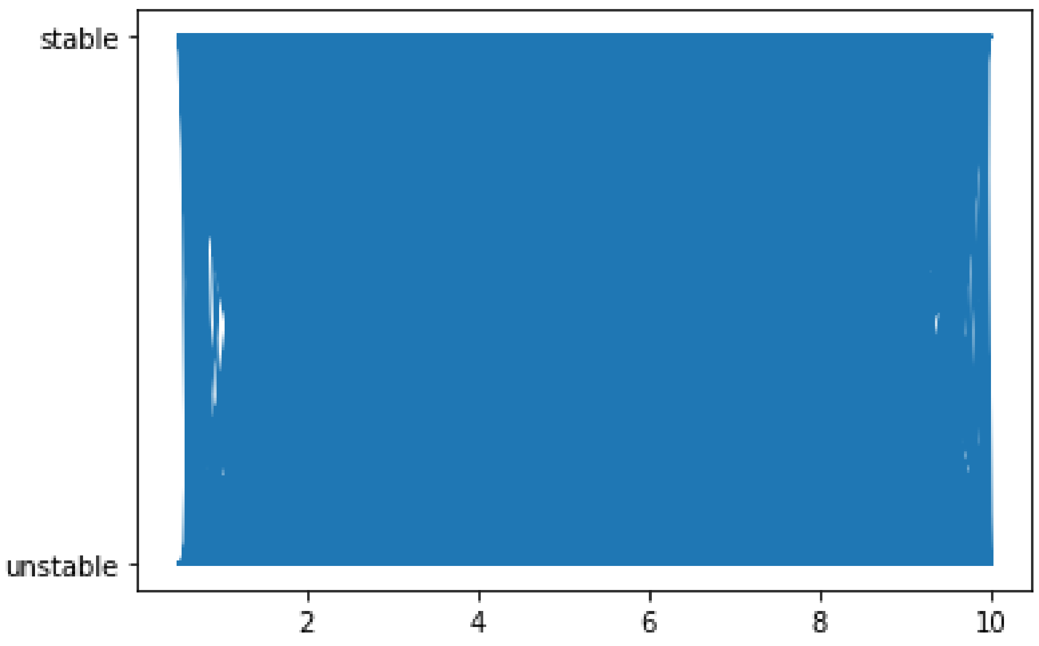
```
[<matplotlib.lines.Line2D  at  0x7f997db267d0>]
```



Out[12]:

```
x4=my_data['tau4']x4=x4.fillna(0)
from matplotlib import pyplot as ptpt.plot(x4,y)
```

```
[<matplotlib.lines.Line2D  at  0x7f997db04910>]
```

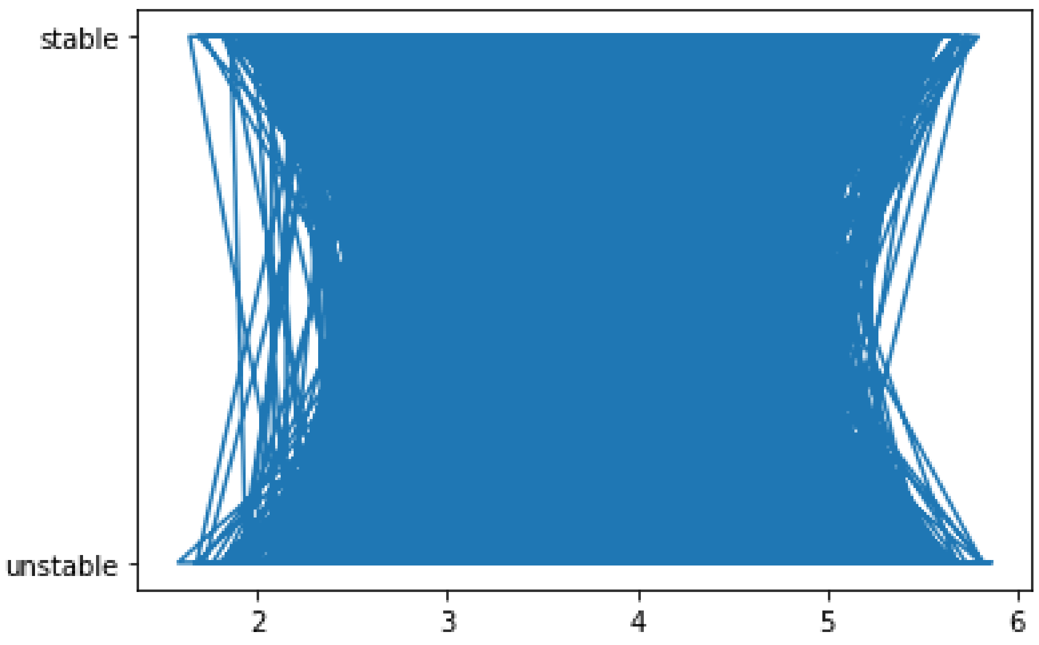


In [13]:

```
x5=my_data['p1']x5=x5.fillna(0)
from matplotlib import pyplot as ptpt.plot(x5,y)
```

Out[13]:

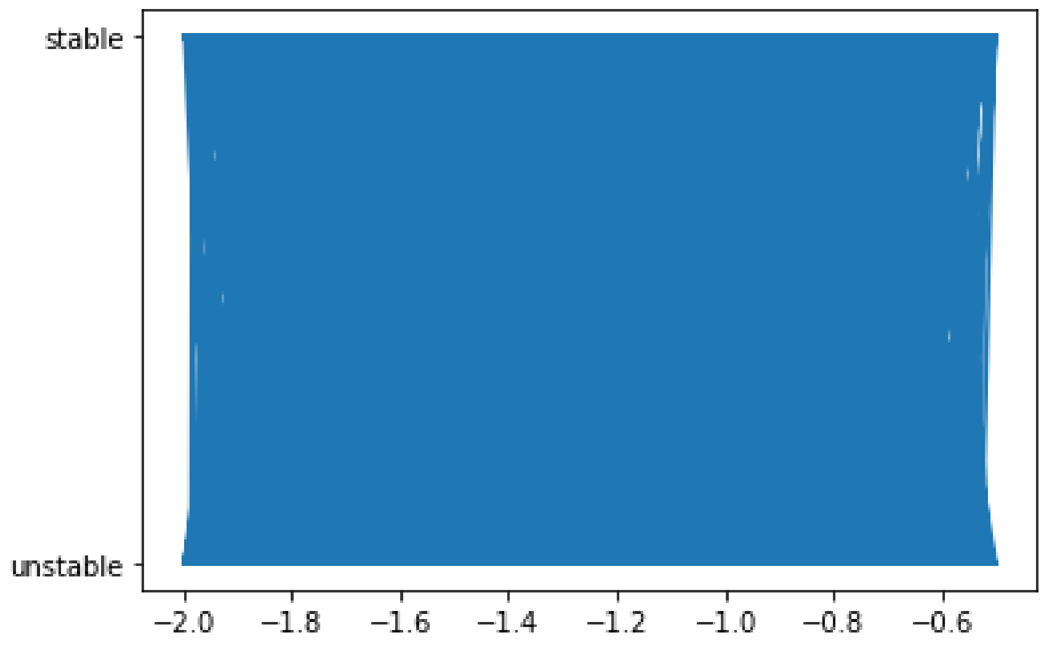
```
[<matplotlib.lines.Line2D  at  0x7f997da22410>]
```



```
x6=my_data['p2']x6=x6.fillna(0)
from matplotlib import pyplot as ptpt.plot(x6,y)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x7f997d9fc410>]
```

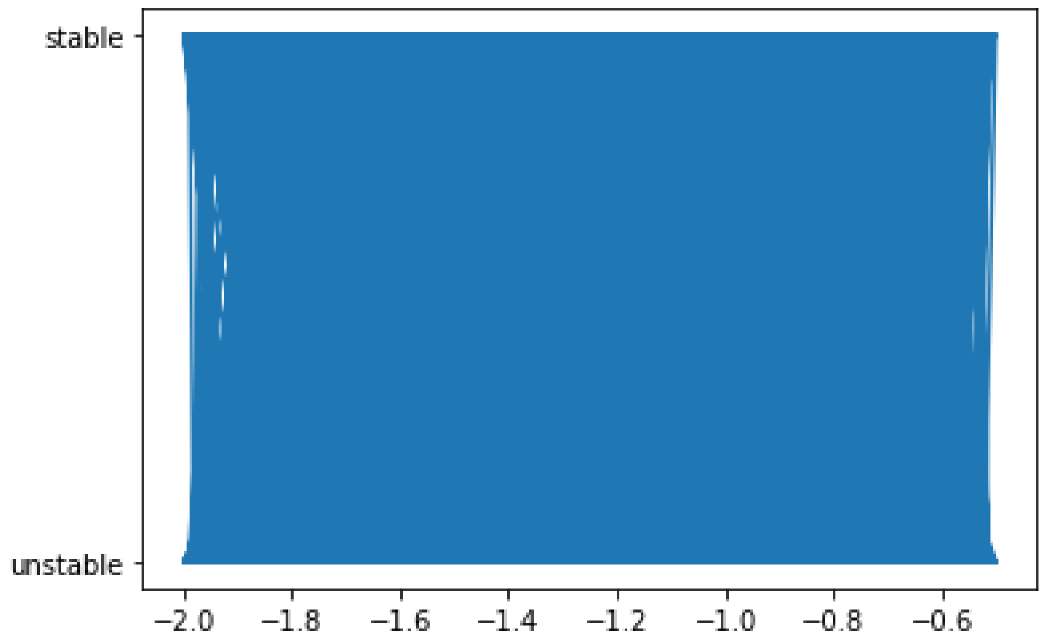


In [15]:

```
x7=my_data['p3']x7=x7.fillna(0)
from matplotlib import pyplot as ptpt.plot(x7,y)
```

Out[15]:

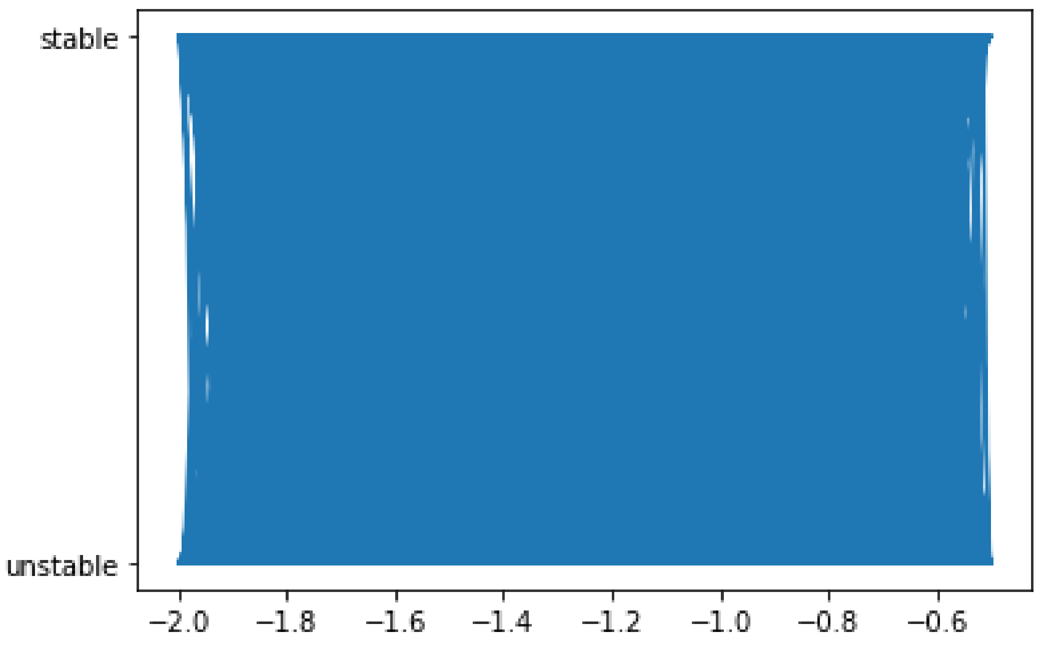
```
[<matplotlib.lines.Line2D  at  0x7f997d9631d0>]
```



```
x8=my_data['p4']x8=x8.fillna(0)
from matplotlib import pyplot as ptpt.plot(x8,y)
```

Out[16]:

```
[<matplotlib.lines.Line2D  at  0x7f997d948510>]
```

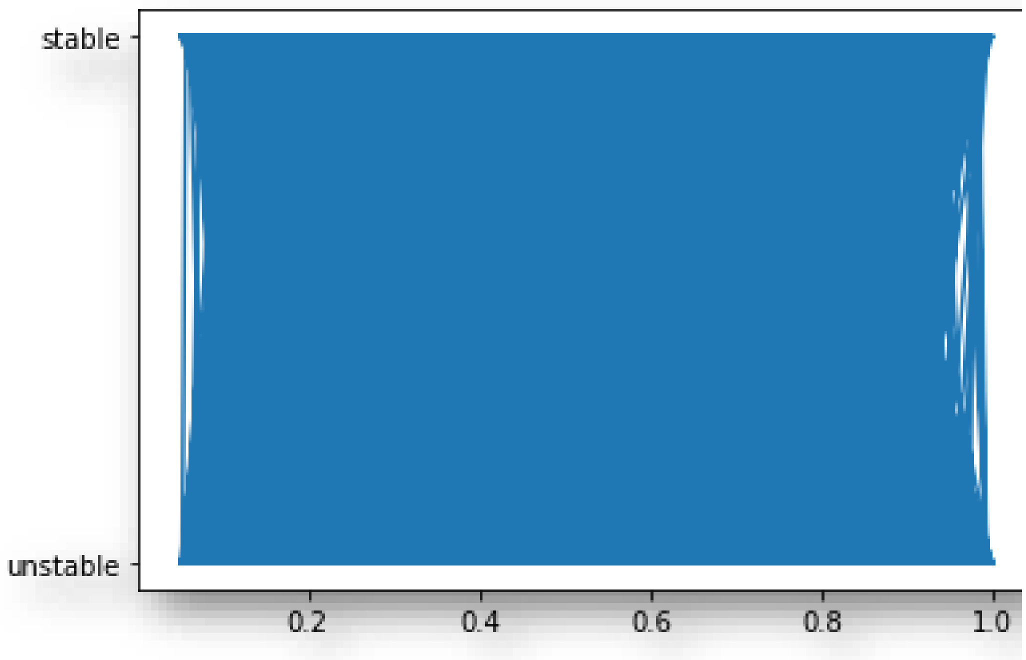


In [17]:

```
x9=my_data['g1']x9=x9.fillna(0)
from matplotlib import pyplot as ptpt.plot(x9,y)
```

Out[17]:

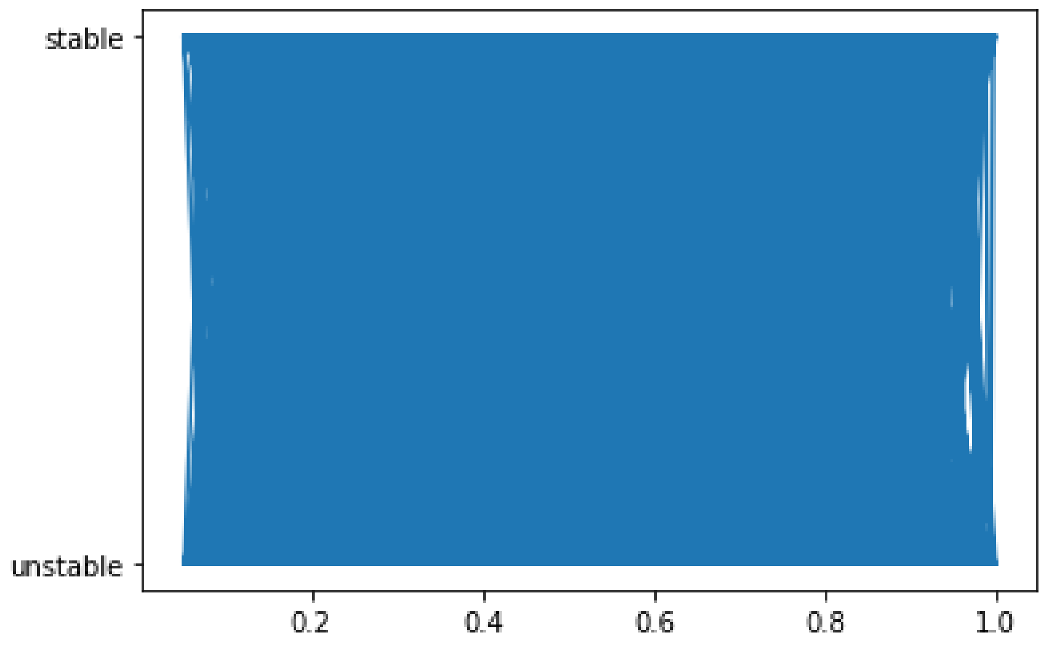
```
[<matplotlib.lines.Line2D  at  0x7f997d8bd1d0>]
```



Out[18]:

```
x10=my_data['g2']x10=x10.fillna(0)
from matplotlib import pyplot as ptpt.plot(x10,y)
```

```
[<matplotlib.lines.Line2D at 0x7f997d894250>]
```

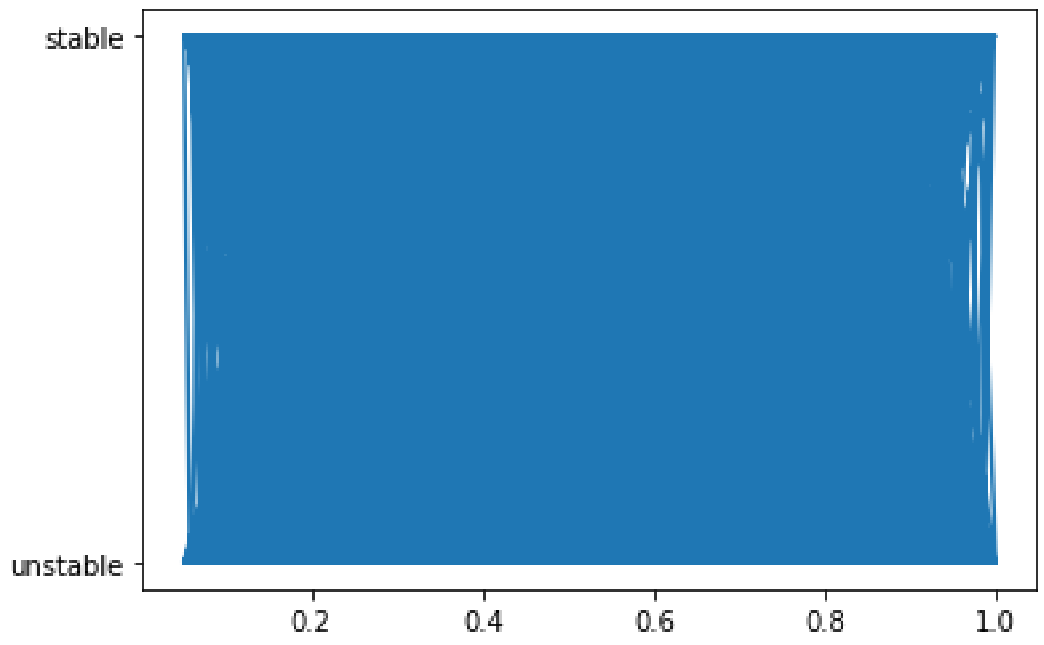


In [19]:

```
x11=my_data['g3']x11=x11.fillna(0)
from matplotlib import pyplot as ptpt.plot(x11,y)
```

Out[19]:

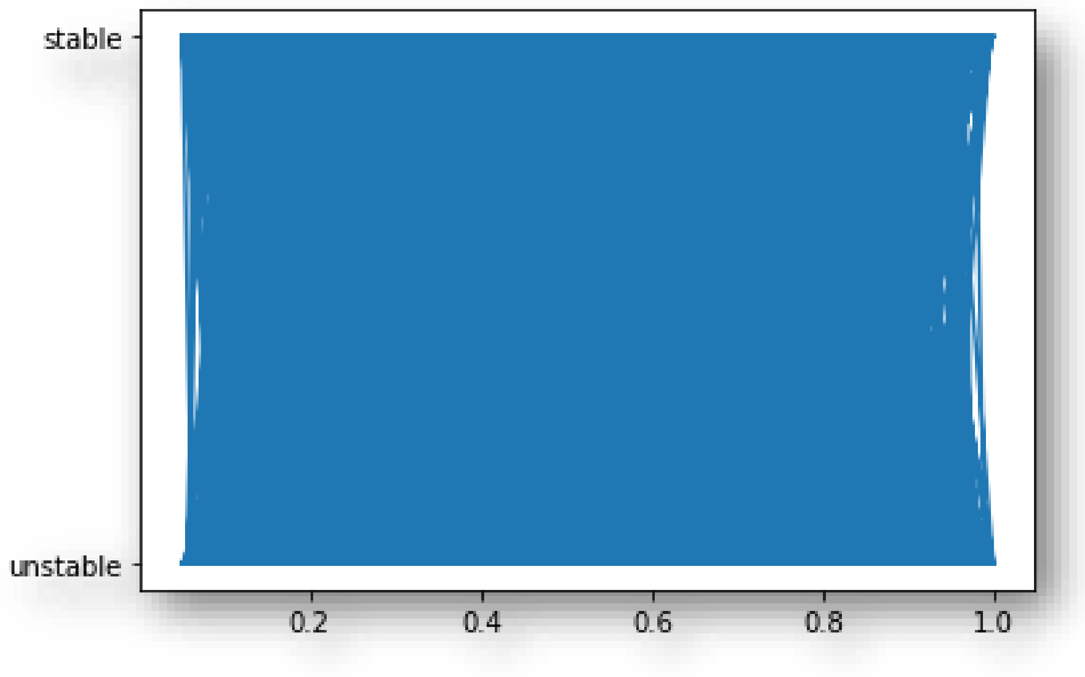
```
[<matplotlib.lines.Line2D at 0x7f997d7ee210>]
```



```
x12=my_data['g4']x12=x12.fillna(0)
from matplotlib import pyplot as ptpt.plot(x12,y)
```

Out[20]:

```
[<matplotlib.lines.Line2D at 0x7f997d7c9250>]
```

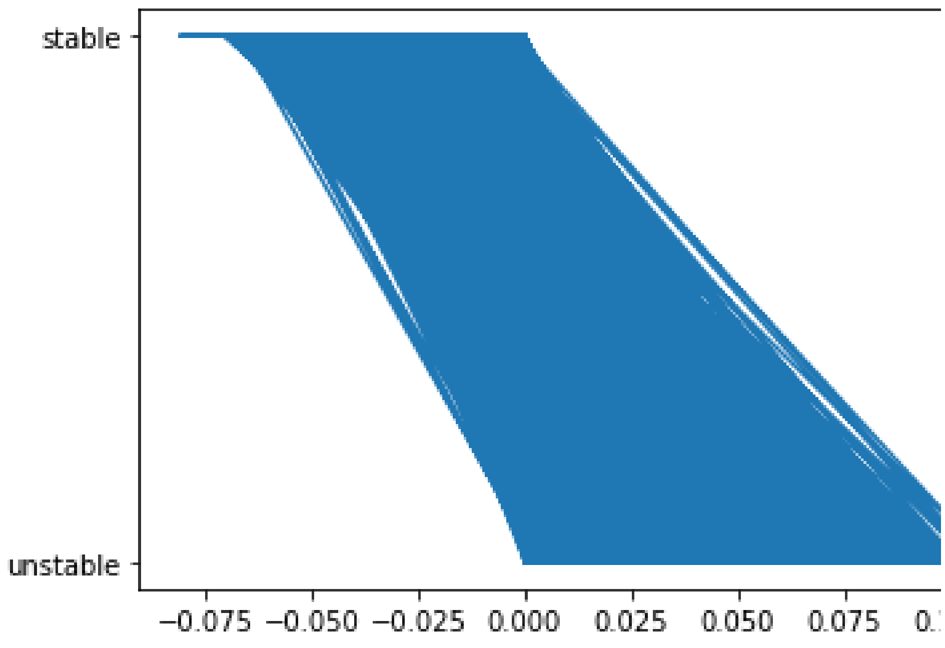


In [21]:

Out[21]:

```
x13=my_data['stab']  x13=x13.fillna(0)
from matplotlib import pyplot as ptpt.plot(x13,y)
```

```
[<matplotlib.lines.Line2D  at  0x7f997d7252d0>]
```



In [ ]:

CHAPTER 5
METHODOLOGY

Enough methods are performed on the data to evaluate the data set and gather knowledge about the data. Let's perform some Machine Learning model and Experimentation to create a model that helps us to achieve our goal we state in the problem definition. In this we talks about the various machine learning algorithms used for the project. They are Multi linear regression, Logistic regression and Lasso regression.

5.1 LOGISTIC REGRESSION

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Logistic Regression is much similar to the gear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

Logistic regression, despite its name, is a classification model rather than regression model. Logistic regression is a simple and more efficient method for binary and linear classification problems. It is a classification model, which is very easy to realize and achieves very good performance with linearly separable classes. It is an extensively employed algorithm for classification in industry. The logistic regression model, like the Adaline and perceptron, is a statistical method for binary classification that can be generalized to multiclass classification.

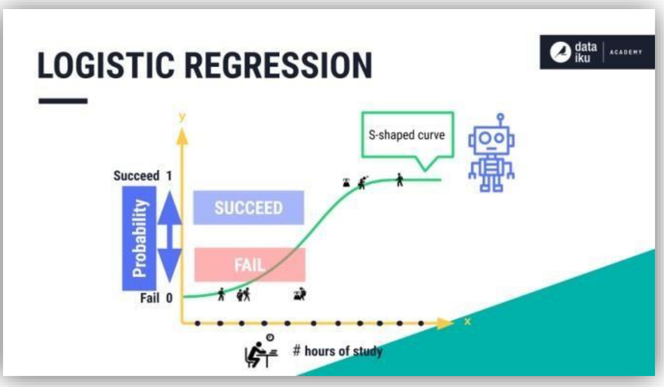


Fig 5.1: Logistic Regression

Formula and Examples of Logistic Regression:

The logistic function is of the form:

$$p(x) = \frac{1}{1 + e^{-(x-\mu)/s}}$$

where μ is a location parameters

is a scale parameter.

Examples where logistic regression may be used

- In Natural Language Processing (NLP), it's used to determine the sentiment of moviereviews.
- In Medicine it can be used to determine the probability of a patient developing a particular disease.
- Predict whether or not a patient will have a heart attack.

5.2 OVERVIEW TECHNOLOGY

In our program we used python 3 programming language .Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typingand dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

TESTING

12 predictive attributes

2 Goal fields:

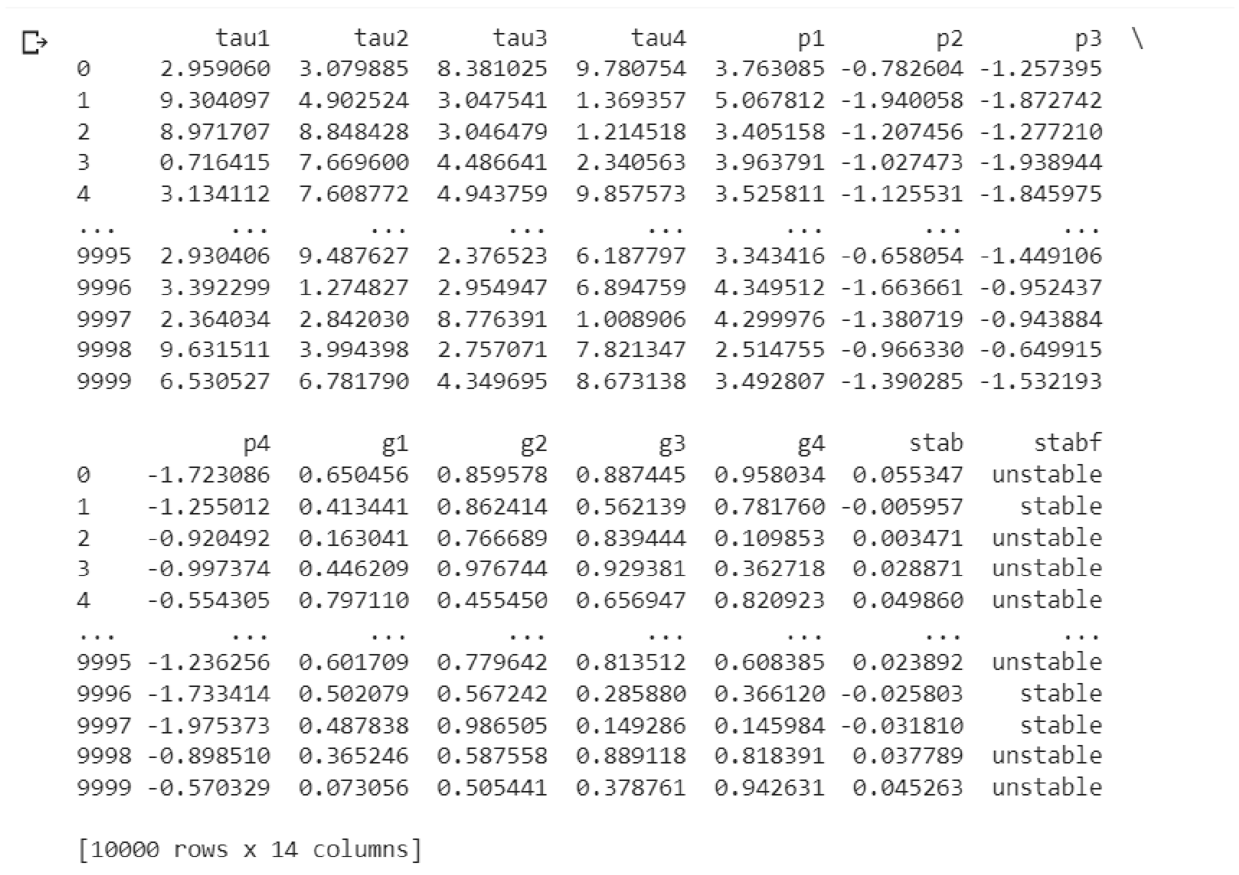
1. tau[x]
2. p[x]
3. g[x]
4. stab
5. stabf

❖ <u>Software Description</u>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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]
```

```
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)
```

```
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
```

```
            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)
```

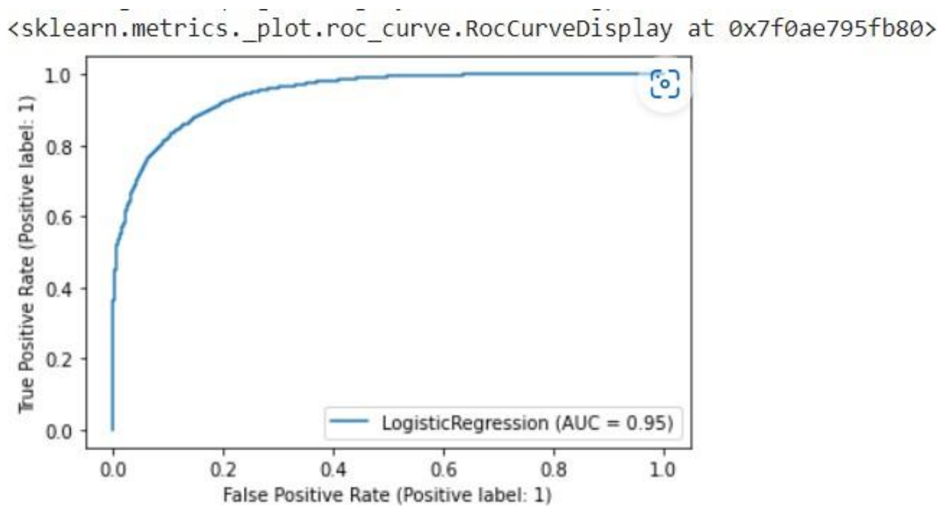
lg=LogisticRegression(random_state=99)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=True)
mm=lg.fit(x_train, y_train)
print('training value',mm.score(x_train,y_train))
print('testing value',mm.score(x_test,y_test))
yp=mm.predict(x_test)
from sklearn.metrics import accuracy_score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

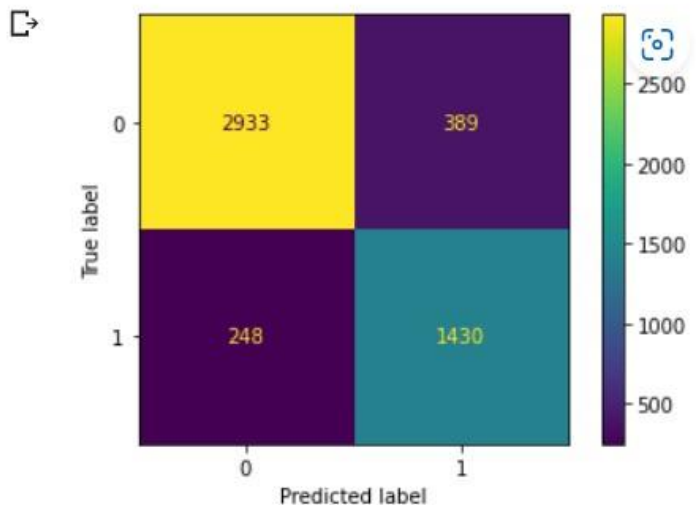
```
              precision    recall  f1-score   support

      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>
```

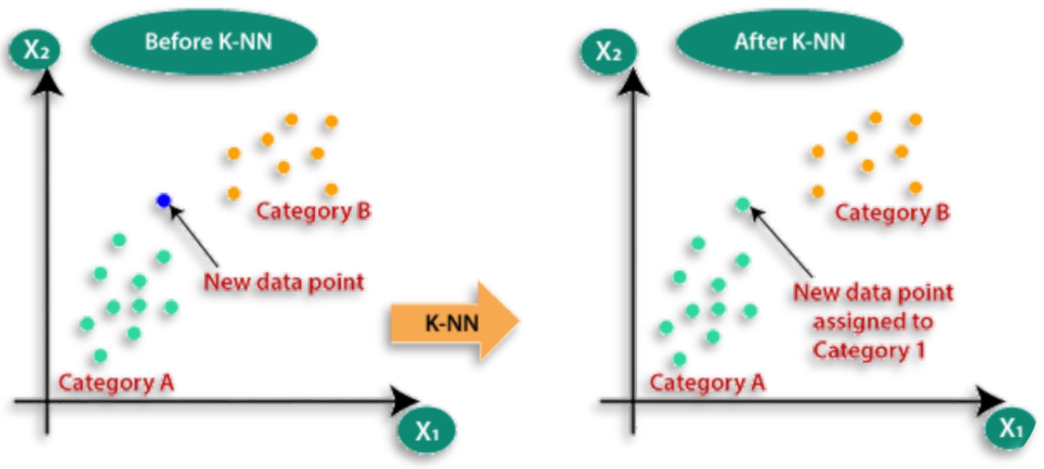
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm=confusion_matrix(yp,y_test)
d=ConfusionMatrixDisplay(cm).plot()



5.3 K-NEAREST NEIGHBORS(K-NN) ALGORITHM

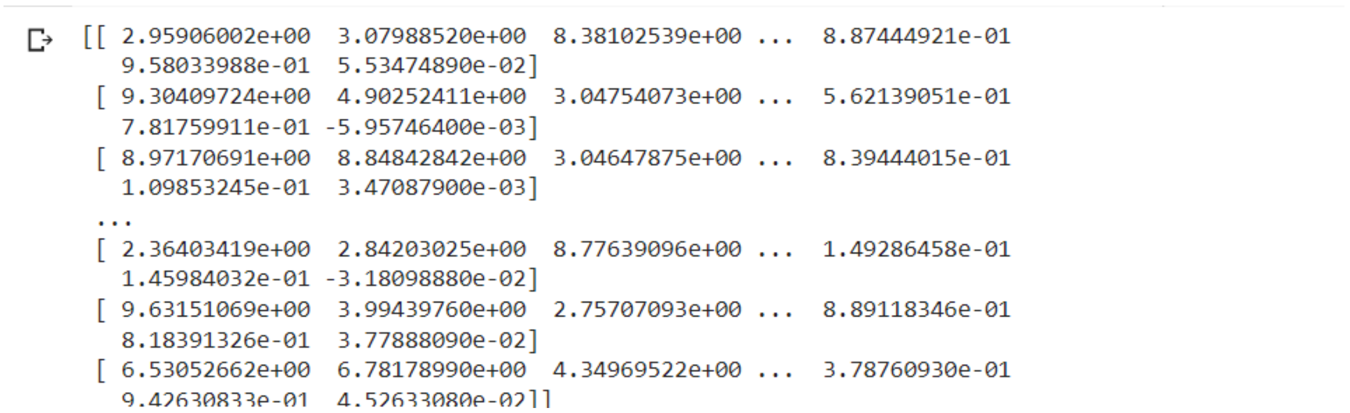
K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

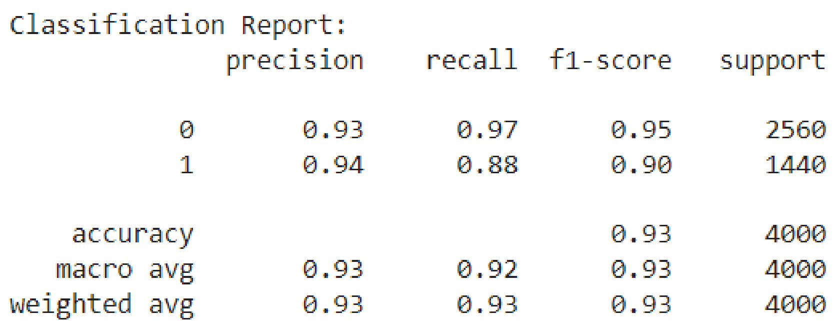


```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('/content/dataset changed.csv')

#Extracting Independent and dependent Variable
X= data_set.iloc[:, 0:13].values
y= data_set.iloc[:, 13:14].values
print(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.40)

# Splitting the dataset into training and test set.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
result = confusion_matrix(y_test, y_pred)
d=ConfusionMatrixDisplay(result).plot()
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(X_train)
x_test= st_x.transform(X_test)
y_train = st_x.fit_transform(y_train)
y_test = st_x.transform(y_test)
```

OUTPUT USING K-NN ALGORITHM

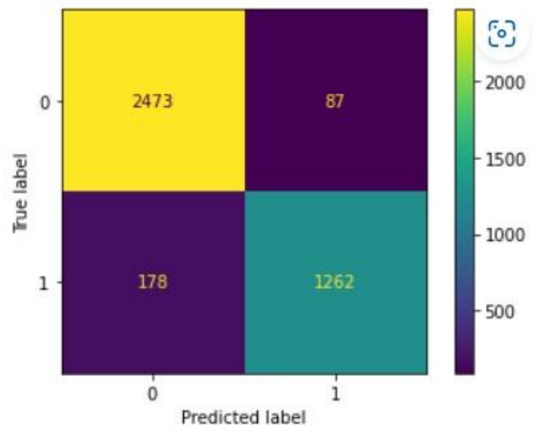


```
[[ 2.95906002e+00  3.07988520e+00  8.38102539e+00 ...  8.87444921e-01
   9.58033988e-01  5.53474890e-02]
 [ 9.30409724e+00  4.90252411e+00  3.04754073e+00 ...  5.62139051e-01
   7.81759911e-01 -5.95746400e-03]
 [ 8.97170691e+00  8.84842842e+00  3.04647875e+00 ...  8.39444015e-01
   1.09853245e-01  3.47087900e-03]
 ...
 [ 2.36403419e+00  2.84203025e+00  8.77639096e+00 ...  1.49286458e-01
   1.45984032e-01 -3.18098880e-02]
 [ 9.63151069e+00  3.99439760e+00  2.75707093e+00 ...  8.89118346e-01
   8.18391326e-01  3.77888090e-02]
 [ 6.53052662e+00  6.78178990e+00  4.34969522e+00 ...  3.78760930e-01
   9.42630833e-01  4.52633080e-02]]
```



```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      2560
           1       0.94      0.88      0.90      1440

    accuracy                           0.93      4000
   macro avg       0.93      0.92      0.93      4000
weighted avg       0.93      0.93      0.93      4000
```

Accuracy: 0.93375



RESULTS:

Algorithm	**Accuracyrate**
Logistic Regression	87.3%
K-Nearest neighbors	93.3%

6.CONCLUSION

Data is being divided in to training and testing dataset.

- Logistic regression gave accuracy score = 87.3%
- KNN as well gave 93.3% accuracy.

So for this dataset, prediction of class is done accurately by KNNand Logistic regression.

7.FUTURE SCOPE

This study considers the evolution of the U.S. electric grid over the next two decades— a period long enough to permit significant change but short enoughto make it unlikely that unforeseen technologies will have significant impacts on the system. I Even though this is likely to be a period of slow growth in the

U.S. demand for electricity by historical standards, public policies and a varietyof technological and economic changes will alter both the demand for and supply of electricity in challenging ways.

8.REFERENCES

- https://rstudio-pubs-static.s3.amazonaws.com/90467_c70206f3dc864d53bf36072207ee011d. html
- https://www.r-bloggers.com/predicting-creditability-using-logistic-regression-in-r-cross-validating-the-classifier-part-2-2/
- "An introduction to Statistical Learning with Applications in R" by Greth James et al
- Arzamasov, Vadim, Klemens BÃ¶hm, and Patrick Jochem. 'Towards Concise Models of Grid Stability.' Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), 2018 IEEEInternational Conference on. IEEE, 2018 (SectionV-A)

Github:- https://github.com/sriramthota1/statml