"OZONE LEVEL DETECTION"

A PROJECT REPORT

Submitted by

SUDESHNA NATH (CSE 3C, 21)
Enrolment Id - 3312016009001706

AVIRUP KUNDU (CSE 3C, 22)
Enrolment Id - 3312016009001048

Data Science & Data Analytics Lab (CS – 695A)

Project Guide - Prof. Sankhadeep Chatterjee & Prof. Somarpita Dutta

B. Tech 3rd year
2019

University of Engineering & Management (UEM)
University Area, Plot No. III - B/5, New Town, Action Area - III, Kolkata, West Bengal 700156

CERTIFICATE

Certified that this project report "**Ozone Level Detection Dataset**" is the bonafide work of

1. Sudeshna Nath (Enrollment no. -3312016009001706)
2. Avirup Kundu (Enrollment no. -3312016009001048)

of B.Tech, CSE, who carried out the project work under our supervision.

………………………………..

………………………………..

SIGNATURE

Examiner:

ACKNOWLEDGEMENT

The completion of this project could not have been accomplished without the support of our teachers and guide Prof. Sankhadeep Chatterjee & Prof. Somarpita Dutta. We are thankful to you for allowing us your time to research and write.

We are also very thankful to our respected teachers for their co-operation and suggestion regarding the project work.

Last but not the least we are very thankful to our HOD, Prof. Sukalyan Goswami for giving us an opportunity of doing such an interesting project work.

- **Sudeshna Nath**
- **Avirup Kundu**

OVERVIEW

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across.

As it is evident from the name, it gives the computer that which makes it more similar to humans: The ability to learn.

Machine learning is actively being used today, perhaps in many more places than one would expect.

The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.

TYPES OF LEARNING:

1. Supervised Learning
2. Unsupervised Learning

INTRODUCTION

Data mining is concerned with locating hidden relationships present in commercial enterprise data presents groups to make predictions for eventual use.

Data mining has risen as a key commercial enterprise intelligence technology. The preference of data mining is to extract implicit, previously unexplored and doubtlessly beneficial (or actionable) styles from statistics.

Data mining encompass many up to date tactics along with classification (neural networks, k-nearest neighbour, naive Bayes classifier, and decision trees), clustering (density based clustering, k-means, hierarchical clustering), association (constraint-based association, multilevel association, multidimensional, one-dimensional).

Years of education display that data mining is a technique, and it's helpful application calls for post processing (presentation, understanding ability, summary), facts pre-processing (cleaning, noise/outlier removal, dimensionality reduction) true knowledge of problem domains and domain facility.

All traditional algorithms are exaggerated to some amount by the class imbalance crisis. Also, the accurate choice of the metric (or consolidation of metrics) to assess – and ultimately improve, is essential for the accomplishment of a data mining effort in such areas, since most of the time improving one metric degrades others.

ALGORITHMS

SUPPORT VECTOR MACHINE (SVM):

A support-vector machine constructs a hyperplane or set of hyperplanes in a high or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

Whereas the original problem may be stated in a finite-dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space.

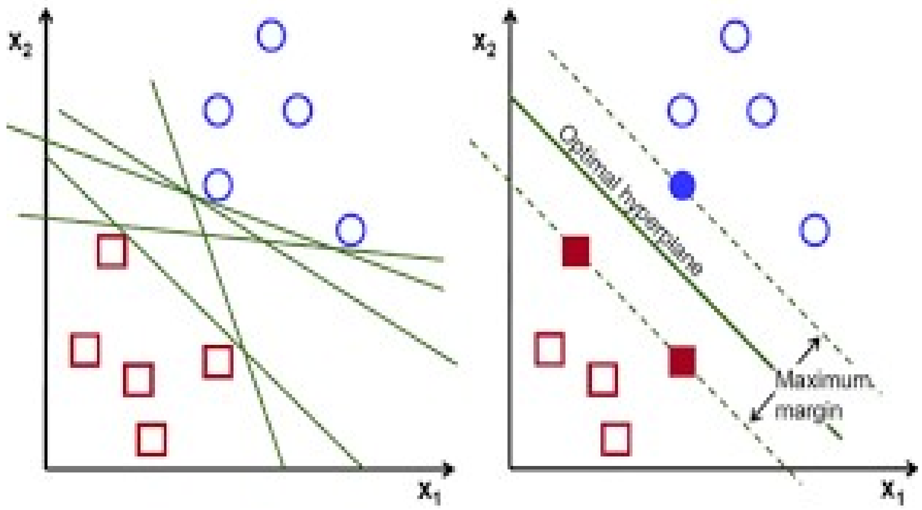
To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function k(x, y) selected to suit the problem.

The hyperplanes in the higher-dimensional space are defined as the set of points whose dot product with a vector in that space is constant, where such a set of vector is an orthogonal (and thus minimal) set of vectors that defines a hyperplane.

The vectors defining the hyperplanes can be chosen to be linear combinations with parameters of images of feature vectors x that occur in the data base. With this choice of a hyperplane, the points x in the feature space that are mapped into the hyperplane are defined by the relation.

Note that if k(x, y) becomes small as y grows further away from x, each term in the sum measures the degree of closeness of the test point x to the corresponding data base point. In this way, the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated.

Note the fact that the set of points x mapped into any hyperplane can be quite convoluted as a result, allowing much more complex discrimination between sets that are not convex at all in the original space.



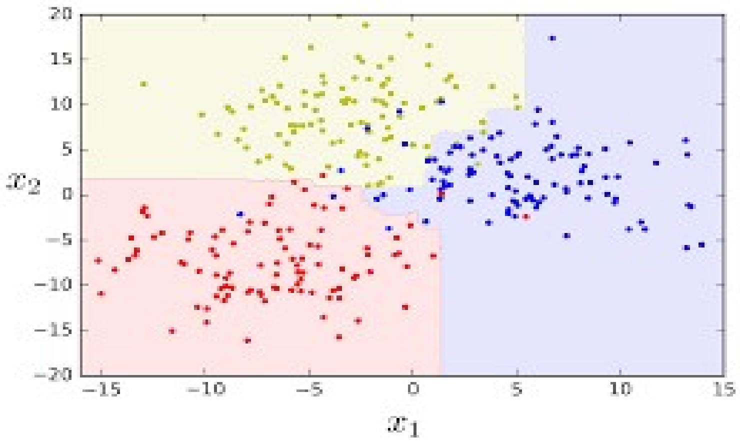
RANDOM FOREST CLASSIFIER:

Decision trees are a popular method for various machine learning tasks. Tree learning *"comes closest to meeting the requirements for serving as an off-the-shelf procedure for data mining*", because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspect able models.

However, they are seldom accurate. In particular, trees that are grown very deep tend to learn highly irregular patterns: they over fit their training sets, i.e. have low bias, but very high variance.

Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.

This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.



K- NEAREST NEIGHBOUR:

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, *k* is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the **overlap metric** (or Hamming distance).

In the context of gene expression microarray data, for example, *k*-NN has also been employed with correlation coefficients such as Pearson and Spearman.

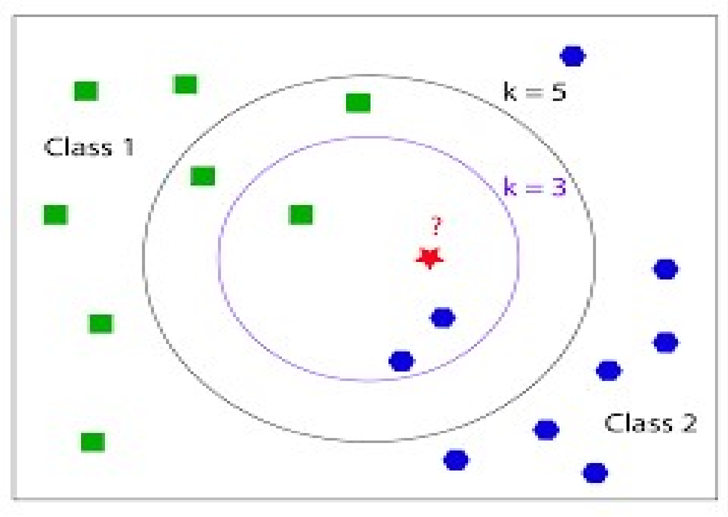
Often, the classification accuracy of *k*-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighborhood components analysis.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the *k* nearest neighbors due to their large number.

One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its *k* nearest neighbors. The class (or value, in regression problems) of each of the *k* nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point.

Another way to overcome skew is by abstraction in data representation.

For example, in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. *K*-NN can then be applied to the SOM.



APPLICATION

- SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labelled training instances in both the standard inductive and transductive settings. Some methods for shallow semantic parsing are based on support vector machines.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. This is also true for image segmentation systems, including those using a modified version SVM that uses the privileged approach as suggested by Vapnik.
- Hand-written characters can be recognized using SVM.
- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models Support-vector machine weights have also been used to interpret SVM models in the past Posthoc interpretation of support-vector machine models in order to identify features used by the model to make predictions is a relatively new area of research with special significance in the biological sciences.

Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}\left|x_i - y_i\right|$$

Minkowski $$\left(\sum_{i=1}^{k}(\left|x_i - y_i\right|)^q\right)^{1/q}$$

SOURCE CODE

1. SUPPORT VECTOR MACHINE

```
import matplotlib.pyplot as plt
from copy import deepcopy
import numpy as np
import pandas as pd
data=pd.read_csv("ozondataset.csv")
data.iloc[:,:]
```

Out[1]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V64	V65	V66	V67	V68	V69	V70	V71	V72
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	0.150000	10.670000	-1.560000	5795.000000	-12.100000	17.900000	10330.000000	-55.000000	0.0
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	0.460000	6.390000	3.840000	5805.000000	14.050000	29.000000	10275.000000	-55.000000	0.0
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	0.600000	6.940000	9.800000	5790.000000	17.900000	41.300000	10235.000000	-40.000000	0.0
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	0.490000	8.730000	10.540000	5775.000000	31.150000	51.700000	10195.000000	-40.000000	2.0
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	0.304716	9.872418	0.830116	5816.821222	10.511051	37.388335	10164.198442	-0.119949	0.5
5	3.1	3.5	3.3	2.5	1.6	1.7	1.6	1.6	2.3	1.8	...	0.090000	11.960000	11.280000	5770.000000	27.950000	46.250000	10120.000000	-0.119949	5.8
6	3.7	3.2	3.8	5.1	6.0	7.0	6.3	6.4	6.3	5.4	...	0.840000	6.880000	25.600000	5695.000000	26.750000	48.450000	10040.000000	-80.000000	0.1
7	2.2	2.9	3.4	4.2	4.7	4.7	5.3	4.9	5.2	6.0	...	0.200000	19.220000	18.210000	5515.000000	-10.100000	42.000000	10065.000000	25.000000	0.0
8	1.0	1.5	1.2	1.2	0.7	0.5	1.2	1.4	1.5	2.1	...	0.510000	9.872418	0.830116	5585.000000	-3.400000	32.900000	10120.000000	-56.000000	0.0
9	0.9	0.6	0.5	0.5	0.8	0.4	0.4	0.6	1.3	1.5	...	0.060000	16.510000	-0.880000	5680.000000	-7.900000	30.500000	10180.000000	60.000000	0.0
10	1.1	1.7	1.4	1.5	0.9	1.5	1.4	1.6	1.9	1.9	...	0.110000	21.770000	0.070000	5715.000000	13.100000	44.700000	10190.000000	10.000000	0.0
11	3.7	4.2	3.1	2.6	2.3	2.3	1.7	1.0	1.3	1.9	...	0.390000	21.070000	5.020000	5740.000000	24.250000	47.850000	10140.000000	-50.000000	0.4
12	1.0	0.6	0.3	1.1	1.3	1.2	1.0	1.3	3.0	2.7	...	0.480000	21.790000	9.140000	5750.000000	7.250000	51.550000	10150.000000	10.000000	0.4
13	1.3	1.3	1.6	1.7	1.4	1.3	1.4	1.4	1.1	2.5	...	0.490000	23.100000	16.880000	5740.000000	29.300000	47.300000	10155.000000	5.000000	0.1
14	4.2	5.1	5.1	5.1	5.5	5.0	5.4	5.6	5.3	5.5	...	0.360000	21.320000	17.220000	5680.000000	20.900000	50.950000	10115.000000	-40.000000	0.0
15	0.0	0.2	0.1	0.2	0.7	0.3	0.3	0.0	1.1	2.6	...	0.360000	26.520000	-7.080000	5585.000000	-9.500000	36.700000	10145.000000	30.000000	0.0
16	2.1	2.2	2.2	1.7	2.1	1.8	1.3	0.8	1.3	3.1	...	0.170000	24.720000	0.980000	5720.000000	1.500000	36.700000	10160.000000	15.000000	0.0
17	2.5	2.3	1.3	1.7	1.6	1.4	2.8	4.3	4.2	5.5	...	0.060000	23.560000	0.360000	5755.000000	1.750000	35.650000	10165.000000	5.000000	0.0
18	2.7	2.0	2.6	2.9	3.3	4.4	3.6	3.2	3.8	3.6	...	0.260000	19.680000	-5.310000	5730.000000	0.500000	33.500000	10150.000000	-15.000000	0.0
19	0.3	0.6	1.1	1.1	1.7	1.9	2.6	3.1	2.5	2.3	...	0.150000	8.210000	-0.540000	5745.000000	-19.800000	16.700000	10175.000000	25.000000	0.0
20	2.1	1.8	1.1	1.5	1.4	1.7	1.5	1.9	1.9	2.9	...	0.020000	16.860000	8.830000	5710.000000	7.600000	39.400000	10130.000000	-45.000000	0.6
21	2.8	3.6	4.1	3.2	3.3	3.7	4.2	4.6	4.6	5.4	...	0.200000	21.110000	12.060000	5690.000000	-30.400000	52.250000	10090.000000	-40.000000	0.0
22	3.4	3.5	3.8	3.2	3.1	4.0	3.4	2.9	2.6	2.1	...	0.320000	17.840000	10.110000	5645.000000	-7.800000	34.700000	10145.000000	55.000000	0.0
2512	0.7	0.2	0.5	0.4	1.3	1.3	1.6	2.2	3.0	4.2	...	0.120000	29.800000	-2.900000	5730.000000	-0.700000	29.800000	10105.000000	-15.000000	0.0
2513	1.2	0.8	0.9	0.9	1.0	0.7	0.8	1.7	1.4	2.6	...	0.160000	26.740000	-16.170000	5715.000000	-21.900000	22.300000	10180.000000	75.000000	0.0
2514	0.9	0.8	0.9	0.5	0.2	0.4	0.8	1.3	1.2	3.4	...	0.260000	11.870000	-11.650000	5785.000000	-1.700000	23.800000	10175.000000	-5.000000	0.0
2515	0.3	0.4	1.3	1.9	3.4	3.6	4.5	4.5	4.6	5.1	...	0.210000	12.830000	-4.670000	5820.000000	2.450000	37.050000	10170.000000	-5.000000	0.0
2516	3.3	3.7	3.7	3.3	2.9	2.7	2.8	2.6	3.2	4.1	...	0.250000	18.130000	-4.120000	5795.000000	-28.700000	13.600000	10305.000000	135.000000	0.0
2517	0.6	0.8	1.1	1.7	1.6	1.9	2.0	2.0	3.2	4.0	...	0.400000	16.770000	-6.350000	5760.000000	-56.700000	0.400000	10350.000000	45.000000	0.0
2518	2.5	2.2	2.2	2.6	2.6	2.3	2.5	2.7	2.8	3.8	...	0.120000	12.520000	0.880000	5745.000000	-16.200000	22.450000	10290.000000	-60.000000	1.1
2519	2.5	3.0	2.8	2.8	2.8	2.4	3.0	3.1	3.0	2.7	...	0.200000	24.220000	1.090000	5730.000000	26.700000	44.150000	10270.000000	-20.000000	0.0
2520	0.3	0.4	0.5	0.5	0.4	0.4	0.2	0.4	0.6	1.7	...	0.390000	17.420000	-0.670000	5695.000000	-18.700000	37.100000	10245.000000	-25.000000	0.0
2521	0.8	0.6	0.7	1.0	1.2	1.2	1.2	2.0	2.8	4.0	...	0.090000	16.990000	-9.390000	5665.000000	-17.100000	30.000000	10225.000000	-20.000000	0.0
2522	0.6	1.1	1.2	1.5	1.4	1.6	1.6	1.4	1.3	3.0	...	0.100000	8.380000	-14.410000	5710.000000	-8.500000	25.200000	10245.000000	20.000000	0.0
2523	1.3	1.1	1.2	1.2	1.4	1.2	0.4	0.3	2.1	2.3	...	0.130000	7.540000	3.140000	5720.000000	-10.850000	41.750000	10165.000000	-80.000000	0.6
2524	0.5	0.5	0.8	0.5	0.5	0.8	0.5	1.6	3.0	5.7	...	0.450000	17.510000	8.930000	5680.000000	22.600000	46.700000	10100.000000	-65.000000	0.4
2525	6.5	6.0	5.6	4.6	4.6	4.9	3.5	3.4	3.2	3.8	...	0.110000	9.872418	0.830116	5670.000000	-11.100000	34.000000	10145.000000	45.000000	0.0
2526	3.6	4.0	3.9	3.8	4.2	4.2	3.6	3.9	3.9	4.3	...	0.190000	30.210000	11.000000	5670.000000	-5.700000	17.000000	10255.000000	110.000000	0.0
2527	2.3	2.3	2.7	2.2	2.4	3.0	2.9	2.4	2.3	2.5	...	0.770000	19.820000	11.830000	5620.000000	7.100000	26.750000	10220.000000	-35.000000	0.0
2528	0.9	0.4	0.3	0.2	0.3	0.2	0.3	0.3	0.3	0.6	...	0.130000	6.770000	-11.980000	5670.000000	-6.600000	25.500000	10230.000000	10.000000	0.0
2529	0.3	0.4	0.5	0.5	0.2	0.3	0.4	0.4	1.3	2.2	...	0.070000	7.930000	-4.410000	5800.000000	-25.600000	21.800000	10295.000000	65.000000	0.0
2530	1.0	1.4	1.1	1.7	1.5	1.7	1.8	1.5	2.1	2.4	...	0.040000	5.950000	-1.140000	5845.000000	-19.400000	19.100000	10310.000000	15.000000	0.0
2531	0.8	0.8	1.2	0.9	0.4	0.6	0.8	1.1	1.5	1.5	...	0.060000	7.800000	-0.840000	5845.000000	-9.600000	35.200000	10275.000000	-35.000000	0.0
2532	1.3	0.9	1.5	1.2	1.6	1.6	1.1	1.0	1.9	2.0	...	0.250000	7.720000	-0.890000	5845.000000	-19.800000	34.200000	10245.000000	-30.000000	0.0
2533	1.5	1.3	1.8	1.4	1.2	1.7	1.6	1.4	1.8	3.0	...	0.540000	13.070000	9.150000	5820.000000	1.950000	39.350000	10220.000000	-25.000000	0.0

2534 rows × 73 columns

```
x=data.iloc[:,1:72].values
y=data["Class"].values
print(y)
```

```
[1 1 1 ... 1 1 1]
```

```
from sklearn.model_selection import train_test_split as tts
x_train,x_test,y_train,y_test=
tts(x,y,test_size=0.40,random_state=42)
#support vector classifier
#test size+train size=1.then We have to train our machine than to
test. so test_size is <.5
#random_state is used to pick trained data randomly
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1520, 71)
(1014, 71)
(1520,)
(1014,)
```

```
from sklearn import svm#support vector machine
cl=svm.SVC(gamma=0.001)
cl.fit(x_train,y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

```
y_predict=cl.predict(x_test)
```

```
from sklearn.metrics import accuracy_score as acc
print(acc(y_test,y_predict))
```

```
0.9349112426035503
```

2. K NEAREST NEIGHBOUR

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
data = pd.read_csv("ozondataset.csv")
print(data.shape)
data.iloc[:,:]
```

```
(2534, 73)
```

Out[1]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V64	V65	V66	V67	V68	V69	V70	V71	V7
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	0.150000	10.670000	-1.560000	5795.000000	-12.100000	17.900000	10330.000000	-55.000000	0.0
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	0.460000	6.390000	3.840000	5805.000000	14.050000	29.000000	10275.000000	-55.000000	0.0
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	0.600000	6.940000	9.800000	5790.000000	17.900000	41.300000	10235.000000	-40.000000	0.0
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	0.490000	8.730000	10.540000	5775.000000	31.150000	51.700000	10195.000000	-40.000000	2.0
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	0.304716	9.872418	0.830116	5818.821222	10.511051	37.388335	10164.198442	-0.119949	0.5
5	3.1	3.5	3.3	2.5	1.6	1.7	1.6	1.6	2.3	1.8	...	0.090000	11.980000	11.280000	5770.000000	27.950000	46.250000	10120.000000	-0.119949	5.8
6	3.7	3.2	3.8	5.1	6.0	7.0	6.3	6.4	6.3	5.4	...	0.840000	6.880000	25.600000	5695.000000	26.750000	48.450000	10040.000000	-80.000000	0.1
7	2.2	2.9	3.4	4.2	4.7	4.7	5.3	4.9	5.2	6.0	...	0.200000	19.220000	18.210000	5515.000000	-10.100000	42.000000	10065.000000	25.000000	0.0
8	1.0	1.5	1.2	1.2	0.7	0.5	1.2	1.4	1.5	2.1	...	0.510000	9.872418	0.830116	5585.000000	-3.400000	32.900000	10120.000000	55.000000	0.0
9	0.9	0.8	0.5	0.5	0.8	0.4	0.4	0.8	1.3	1.5	...	0.060000	16.510000	-0.880000	5680.000000	-7.900000	30.500000	10180.000000	60.000000	0.0
10	1.1	1.7	1.4	1.5	0.9	1.5	1.4	1.6	1.9	1.9	...	0.110000	21.770000	0.070000	5715.000000	13.100000	44.700000	10190.000000	10.000000	0.0
11	3.7	4.2	3.1	2.8	2.3	2.3	1.7	1.0	1.3	1.9	...	0.390000	21.070000	5.020000	5740.000000	24.250000	47.850000	10140.000000	-50.000000	0.4
12	1.0	0.6	0.3	1.1	1.3	1.2	1.0	1.3	3.0	2.7	...	0.480000	21.790000	9.140000	5750.000000	7.250000	51.550000	10150.000000	10.000000	0.4
13	1.3	1.3	1.6	1.7	1.4	1.3	1.4	1.4	1.1	2.5	...	0.490000	23.100000	16.880000	5740.000000	29.300000	47.300000	10155.000000	5.000000	0.1
14	4.2	5.1	5.1	5.1	5.5	5.0	5.4	5.6	5.3	5.5	...	0.360000	21.320000	17.220000	5660.000000	20.900000	50.950000	10115.000000	-40.000000	0.0
15	0.0	0.2	0.1	0.2	0.7	0.3	0.3	0.0	1.1	2.6	...	0.360000	26.520000	-7.080000	5585.000000	-9.500000	36.700000	10145.000000	30.000000	0.0
16	2.1	2.2	2.2	1.7	2.1	1.8	1.3	0.8	1.3	3.1	...	0.170000	24.720000	0.980000	5720.000000	1.500000	36.700000	10160.000000	15.000000	0.0
17	2.5	2.3	1.3	1.7	1.8	1.4	2.8	4.3	4.2	5.5	...	0.060000	23.560000	0.360000	5755.000000	1.750000	35.650000	10165.000000	5.000000	0.0
18	2.7	2.0	2.6	2.9	3.3	4.4	3.6	3.2	3.8	3.6	...	0.260000	19.680000	-5.310000	5730.000000	0.500000	33.500000	10150.000000	-15.000000	0.0
19	0.3	0.6	1.1	1.1	1.7	1.9	2.6	3.1	2.5	2.3	...	0.150000	8.210000	-0.540000	5745.000000	-19.800000	16.700000	10175.000000	25.000000	0.0
20	2.1	1.8	1.1	1.5	1.4	1.7	1.5	1.9	1.9	2.9	...	0.020000	16.860000	8.830000	5710.000000	7.600000	39.400000	10130.000000	-45.000000	0.6
21	2.8	3.6	4.1	3.2	3.3	3.7	4.2	4.6	4.6	5.4	...	0.200000	21.110000	12.060000	5690.000000	30.400000	52.250000	10090.000000	-40.000000	0.0
22	3.4	3.5	3.8	3.2	3.1	4.0	3.4	2.9	2.6	2.1	...	0.320000	17.840000	10.110000	5645.000000	-7.800000	34.700000	10145.000000	55.000000	0.0
2512	0.7	0.2	0.5	0.4	1.3	1.3	1.6	2.2	3.0	4.2	...	0.120000	29.800000	-2.900000	5730.000000	-0.700000	29.800000	10105.000000	-15.000000	0.0
2513	1.2	0.8	0.9	0.9	1.0	0.7	0.8	1.7	1.4	2.6	...	0.160000	26.740000	-16.170000	5715.000000	-21.900000	22.300000	10180.000000	75.000000	0.0
2514	0.9	0.8	0.9	0.5	0.2	0.4	0.8	1.3	1.2	3.4	...	0.260000	11.870000	-11.650000	5785.000000	-1.700000	23.800000	10175.000000	-5.000000	0.0
2515	0.3	0.4	1.3	1.9	3.4	3.6	4.5	4.5	4.6	5.1	...	0.210000	12.830000	-4.670000	5620.000000	2.450000	37.050000	10170.000000	-5.000000	0.0
2516	3.3	3.7	3.7	3.3	2.9	2.7	2.8	2.8	3.2	4.1	...	0.250000	18.130000	-4.120000	5795.000000	-28.700000	13.600000	10305.000000	135.000000	0.0
2517	0.6	0.8	1.1	1.7	1.8	1.9	2.0	2.0	3.2	4.0	...	0.400000	16.770000	-6.350000	5760.000000	-56.700000	0.400000	10350.000000	45.000000	0.0
2518	2.5	2.2	2.2	2.6	2.6	2.3	2.5	2.7	2.8	3.8	...	0.120000	12.520000	0.860000	5745.000000	-16.200000	22.450000	10290.000000	-60.000000	1.1
2519	2.5	3.0	2.6	2.6	2.6	2.4	3.0	3.1	3.0	2.7	...	0.200000	24.220000	1.090000	5730.000000	26.700000	44.150000	10270.000000	-20.000000	0.0
2520	0.3	0.4	0.5	0.5	0.4	0.4	0.2	0.4	0.6	1.7	...	0.390000	17.420000	-0.670000	5695.000000	-18.700000	37.100000	10245.000000	-25.000000	0.0
2521	0.8	0.6	0.7	1.0	1.2	1.2	1.2	2.0	2.8	4.0	...	0.090000	16.990000	-9.390000	5665.000000	-17.100000	30.000000	10225.000000	-20.000000	0.0
2522	0.6	1.1	1.2	1.5	1.4	1.6	1.6	1.4	1.3	3.0	...	0.100000	8.380000	-14.410000	5710.000000	-8.500000	25.200000	10245.000000	20.000000	0.0
2523	1.3	1.1	1.2	1.2	1.4	1.2	0.4	0.3	2.1	2.3	...	0.130000	7.540000	3.140000	5720.000000	-10.850000	41.750000	10165.000000	-80.000000	0.6
2524	0.5	0.5	0.8	0.5	0.5	0.8	0.5	1.6	3.0	5.7	...	0.450000	17.510000	8.930000	5680.000000	22.600000	46.700000	10100.000000	-65.000000	0.4
2525	6.5	6.0	5.6	4.6	4.6	4.9	3.5	3.4	3.2	3.8	...	0.110000	9.872418	0.830116	5670.000000	-11.100000	34.000000	10145.000000	45.000000	0.0
2526	3.6	4.0	3.9	3.8	4.2	4.2	3.6	3.9	3.9	4.3	...	0.190000	30.210000	11.000000	5670.000000	-5.700000	17.000000	10255.000000	110.000000	0.0
2527	2.3	2.3	2.7	2.2	2.4	3.0	2.9	2.4	2.3	2.5	...	0.770000	19.820000	11.830000	5620.000000	7.100000	26.750000	10220.000000	-35.000000	0.0
2528	0.9	0.4	0.3	0.2	0.3	0.2	0.3	0.3	0.3	0.6	...	0.130000	6.770000	-11.980000	5670.000000	-6.600000	25.500000	10230.000000	10.000000	0.0
2529	0.3	0.4	0.5	0.5	0.2	0.3	0.4	0.4	1.3	2.2	...	0.070000	7.930000	-4.410000	5800.000000	-25.600000	21.800000	10295.000000	65.000000	0.0
2530	1.0	1.4	1.1	1.7	1.5	1.7	1.8	1.5	2.1	2.4	...	0.040000	5.950000	-1.140000	5845.000000	-19.400000	19.100000	10310.000000	15.000000	0.0
2531	0.8	0.8	1.2	0.9	0.4	0.6	0.8	1.1	1.5	1.5	...	0.080000	7.800000	-0.840000	5845.000000	-9.600000	35.200000	10275.000000	-35.000000	0.0
2532	1.3	0.9	1.5	1.2	1.6	1.6	1.1	1.0	1.9	2.0	...	0.250000	7.720000	-0.890000	5845.000000	-19.800000	34.200000	10245.000000	-30.000000	0.0
2533	1.5	1.3	1.8	1.4	1.2	1.7	1.8	1.4	1.8	3.0	...	0.540000	13.070000	9.150000	5820.000000	1.950000	39.350000	10220.000000	-25.000000	0.0

2534 rows × 73 columns

```
x1=data.iloc[:,1:72].values
y=data["Class"].values
print(y)
```

```
[1 1 1 ... 1 1 1]
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x1,
y,test_size=0.4)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(x_train, y_train)
```

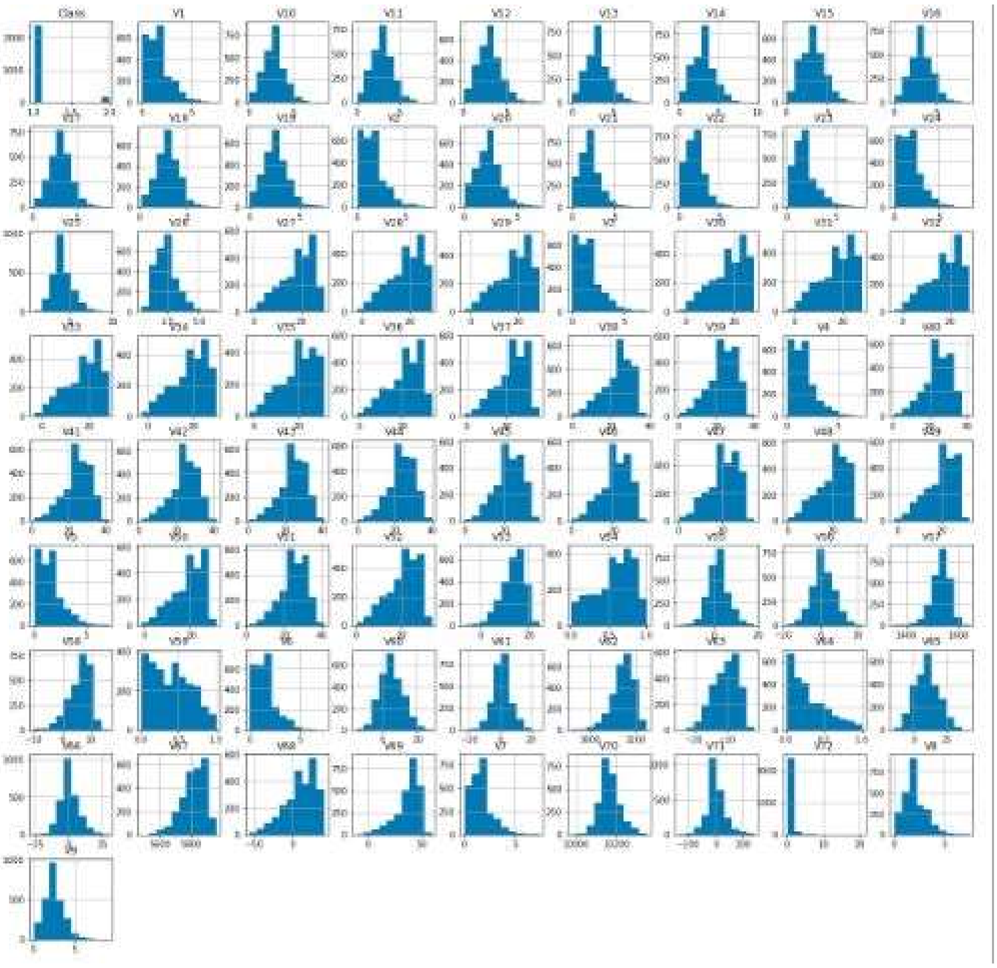
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski'
,
           metric_params=None, n_jobs=None, n_neighbors=6, p=2,
           weights='uniform')
```

```
y_predict=knn.predict(x_test)
```

```
from sklearn.metrics import accuracy_score as acc
print(acc(y_test,y_predict))
```

```
0.9447731755424064
```

```
data.hist(figsize = (20,20))
plt.show
```



3. RANDOM FOREST CLASSIFIER

```
import matplotlib.pyplot as plt
from copy import deepcopy
import numpy as np
import pandas as pd
data=pd.read_csv("ozondataset.csv")
data.iloc[:,:]
```

Out[1]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V64	V65	V66	V67	V68	V69	V70	V71	V72
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	0.150000	10.670000	-1.560000	5795.000000	-12.100000	17.900000	10330.000000	-55.000000	0.0
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	0.460000	6.390000	3.840000	5805.000000	14.050000	29.000000	10275.000000	-55.000000	0.0
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	0.600000	6.940000	9.800000	5790.000000	17.900000	41.300000	10235.000000	-40.000000	0.0
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	0.490000	8.730000	10.540000	5775.000000	31.150000	51.700000	10195.000000	-40.000000	2.0
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	0.304716	9.872418	0.830116	5816.821222	10.511051	37.388335	10164.198442	-0.119949	0.5
5	3.1	3.5	3.3	2.5	1.6	1.7	1.6	1.6	2.3	1.8	...	0.090000	11.980000	11.280000	5770.000000	27.950000	46.250000	10120.000000	-0.119949	5.8
6	3.7	3.2	3.8	5.1	6.0	7.0	6.3	6.4	6.3	5.4	...	0.840000	6.860000	25.600000	5695.000000	26.750000	48.450000	10040.000000	-80.000000	0.1
7	2.2	2.9	3.4	4.2	4.7	4.7	5.3	4.9	5.2	6.0	...	0.200000	19.220000	18.210000	5515.000000	-10.100000	42.000000	10065.000000	25.000000	0.0
8	1.0	1.5	1.2	1.2	0.7	0.5	1.2	1.4	1.5	2.1	...	0.510000	9.872418	0.830116	5585.000000	-3.400000	32.900000	10120.000000	-55.000000	0.0
9	0.9	0.6	0.5	0.5	0.8	0.4	0.4	0.6	1.3	1.5	...	0.060000	16.510000	-0.880000	5680.000000	-7.900000	30.500000	10180.000000	60.000000	0.0
10	1.1	1.7	1.4	1.5	0.9	1.5	1.4	1.6	1.9	1.9	...	0.110000	21.770000	0.070000	5715.000000	13.100000	44.700000	10190.000000	10.000000	0.0
11	3.7	4.2	3.1	2.6	2.3	2.3	1.7	1.0	1.3	1.9	...	0.390000	21.070000	5.020000	5740.000000	24.250000	47.850000	10140.000000	-50.000000	0.4
12	1.0	0.6	0.3	1.1	1.3	1.2	1.0	1.3	3.0	2.7	...	0.480000	21.790000	9.140000	5750.000000	7.250000	51.550000	10150.000000	10.000000	0.4
13	1.3	1.3	1.6	1.7	1.4	1.3	1.4	1.4	1.1	2.5	...	0.490000	23.100000	16.880000	5740.000000	29.300000	47.300000	10155.000000	5.000000	0.1
14	4.2	5.1	5.1	5.1	5.5	5.0	5.4	5.6	5.3	5.5	...	0.360000	21.320000	17.220000	5660.000000	20.900000	50.950000	10115.000000	-40.000000	0.0
15	0.0	0.2	0.1	0.2	0.7	0.3	0.3	0.0	1.1	2.6	...	0.360000	26.520000	-7.080000	5585.000000	-9.500000	36.700000	10145.000000	30.000000	0.0
16	2.1	2.2	2.2	1.7	2.1	1.8	1.3	0.8	1.3	3.1	...	0.170000	24.720000	0.980000	5720.000000	1.500000	36.700000	10160.000000	15.000000	0.0
17	2.5	2.3	1.3	1.7	1.6	1.4	2.8	4.3	4.2	5.5	...	0.060000	23.560000	0.360000	5755.000000	1.750000	35.650000	10165.000000	5.000000	0.0
18	2.7	2.0	2.6	2.9	3.3	4.4	3.6	3.2	3.8	3.6	...	0.260000	19.680000	-5.310000	5730.000000	0.500000	33.500000	10150.000000	-15.000000	0.0
19	0.3	0.6	1.1	1.1	1.7	1.9	2.6	3.1	2.5	2.3	...	0.150000	8.210000	-0.540000	5745.000000	-19.800000	16.700000	10175.000000	25.000000	0.0
20	2.1	1.8	1.1	1.5	1.4	1.7	1.5	1.9	1.9	2.9	...	0.020000	16.860000	8.830000	5710.000000	7.600000	39.400000	10130.000000	-45.000000	0.6
21	2.8	3.6	4.1	3.2	3.3	3.7	4.2	4.6	4.6	5.4	...	0.200000	21.110000	12.060000	5690.000000	-30.400000	52.250000	10090.000000	-40.000000	0.0
22	3.4	3.5	3.8	3.2	3.1	4.0	3.4	2.9	2.6	2.1	...	0.320000	17.640000	10.110000	5645.000000	-7.800000	34.700000	10145.000000	55.000000	0.0
2512	0.7	0.2	0.5	0.4	1.3	1.3	1.8	2.2	3.0	4.2	...	0.120000	29.800000	-2.900000	5730.000000	-0.700000	29.800000	10105.000000	-15.000000	0.0
2513	1.2	0.8	0.9	0.9	1.0	0.7	0.8	1.7	1.4	2.6	...	0.160000	26.740000	-16.170000	5715.000000	-21.900000	22.300000	10180.000000	75.000000	0.0
2514	0.9	0.8	0.9	0.5	0.2	0.4	0.8	1.3	1.2	3.4	...	0.260000	11.870000	-11.650000	5785.000000	-1.700000	23.800000	10175.000000	-5.000000	0.0
2515	0.3	0.4	1.3	1.9	3.4	3.6	4.5	4.5	4.6	5.1	...	0.210000	12.830000	-4.670000	5820.000000	2.450000	37.050000	10170.000000	-5.000000	0.0
2516	3.3	3.7	3.7	3.3	2.9	2.7	2.8	2.6	3.2	4.1	...	0.250000	18.130000	-4.120000	5795.000000	-28.700000	13.600000	10305.000000	135.000000	0.0
2517	0.6	0.8	1.1	1.7	1.6	1.9	2.0	2.0	3.2	4.0	...	0.400000	16.770000	-6.350000	5760.000000	-56.700000	0.400000	10350.000000	45.000000	0.0
2518	2.5	2.2	2.2	2.6	2.6	2.3	2.5	2.7	2.8	3.8	...	0.120000	12.520000	0.880000	5745.000000	-16.200000	22.450000	10290.000000	-60.000000	1.1
2519	2.5	3.0	2.8	2.8	2.6	2.4	3.0	3.1	3.0	2.7	...	0.200000	24.220000	1.090000	5730.000000	26.700000	44.150000	10270.000000	-20.000000	0.0
2520	0.3	0.4	0.5	0.5	0.4	0.4	0.2	0.4	0.6	1.7	...	0.390000	17.420000	-0.670000	5695.000000	-18.700000	37.100000	10245.000000	-25.000000	0.0
2521	0.8	0.6	0.7	1.0	1.2	1.2	1.2	2.0	2.8	4.0	...	0.090000	16.990000	-9.390000	5665.000000	-17.100000	30.000000	10225.000000	-20.000000	0.0
2522	0.6	1.1	1.2	1.5	1.4	1.6	1.6	1.4	1.3	3.0	...	0.100000	8.380000	-14.410000	5710.000000	-8.500000	25.200000	10245.000000	20.000000	0.0
2523	1.3	1.1	1.2	1.2	1.4	1.2	0.4	0.3	2.1	2.3	...	0.130000	7.540000	3.140000	5720.000000	-10.850000	41.750000	10165.000000	-80.000000	0.6
2524	0.5	0.5	0.8	0.5	0.5	0.8	0.5	1.6	3.0	5.7	...	0.450000	17.510000	6.930000	5680.000000	22.600000	46.700000	10100.000000	-65.000000	0.4
2525	6.5	6.0	5.6	4.6	4.6	4.9	3.5	3.4	3.2	3.8	...	0.110000	9.872418	0.830116	5670.000000	-11.100000	34.000000	10145.000000	45.000000	0.0
2526	3.6	4.0	3.9	3.8	4.2	4.2	3.6	3.9	3.9	4.3	...	0.190000	30.210000	11.000000	5670.000000	-5.700000	17.000000	10255.000000	110.000000	0.0
2527	2.3	2.3	2.7	2.2	2.4	3.0	2.9	2.4	2.3	2.5	...	0.770000	19.820000	11.830000	5620.000000	7.100000	26.750000	10220.000000	-35.000000	0.0
2528	0.9	0.4	0.3	0.2	0.3	0.2	0.3	0.3	0.3	0.6	...	0.130000	8.770000	-11.980000	5670.000000	-6.600000	25.500000	10230.000000	10.000000	0.0
2529	0.3	0.4	0.5	0.5	0.2	0.3	0.4	0.4	1.3	2.2	...	0.070000	7.930000	-4.410000	5800.000000	-25.600000	21.800000	10295.000000	65.000000	0.0
2530	1.0	1.4	1.1	1.7	1.5	1.7	1.8	1.5	2.1	2.4	...	0.040000	5.950000	-1.140000	5845.000000	-19.400000	19.100000	10310.000000	15.000000	0.0
2531	0.8	0.8	1.2	0.9	0.4	0.6	0.8	1.1	1.5	1.5	...	0.060000	7.800000	-0.640000	5845.000000	-9.600000	35.200000	10275.000000	-35.000000	0.0
2532	1.3	0.9	1.5	1.2	1.6	1.6	1.1	1.0	1.9	2.0	...	0.250000	7.720000	-0.890000	5645.000000	-19.800000	34.200000	10245.000000	-30.000000	0.0
2533	1.5	1.3	1.8	1.4	1.2	1.7	1.6	1.4	1.8	3.0	...	0.540000	13.070000	9.150000	5820.000000	1.950000	39.350000	10220.000000	-25.000000	0.0

2534 rows × 73 columns

```
x=data.iloc[:,1:72].values
y=data["Class"].values
print(y)
```

```
[1 1 1 ... 1 1 1]
```

```
from sklearn.model_selection import train_test_split as tts
x_train,x_test,y_train,y_test=
tts(x,y,test_size=0.40,random_state=42)
#support vector classifier
#test size+train size=1.then We have to train our machine than to
test. so test_size is <.5
#random_state is used to pick trained data randomly
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1520, 71)
(1014, 71)
(1520,)
(1014,)
```

```
from sklearn.ensemble import RandomForestClassifier
cl=RandomForestClassifier(n_estimators=4)
cl.fit(x_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=4, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
y_predict=cl.predict(x_test)

from sklearn.metrics import accuracy_score as acc
print(acc(y_test,y_predict))
```

```
0.9398422090729783
```

CONCLUSION

After successful completion of this project we can conclude that Ozone Level Detection Dataset can be used in data science for good purpose.

Here we have used Support Vector Machine, Random Forest, K-Nearest Neighbour Classifier for analysis of Ozone Level Detection.

In case of SVM we got an accuracy of 0.9349112426035503

In case of KNN classifier we got an accuracy of 0.9447731755424064

In case of Random forest we got an accuracy of 0.9398422090729783