UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CSE)

SEM/YEAR: 6th Semester / 3rd Year

SECTION: CSE-3C

SUBJECT NAME: Data Science & Data Analytics Lab

SUBJECT CODE: CS695A

PROJECT: OZONE LEVEL DETECTION

GROUP MEMBERS:

Name	Roll	Enrollment no.	Reg. no.
ARPAN DUTTA	21	12017009001313	304201700900161
SURJENDU DAS	22	12017009001229	304201700900805

CERTIFICATE

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

1. Arpan Dutta (Enrollment no. -33120170090013 13)
2. Surjendu Das (Enrollment no. -33120170090012 29)

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. Moumita Basu. 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.

- **Arpan Dutta**
- **Surjendu Das**

PROJECT REPORT

INTRODUCTION:

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. As it is evident from the name, it gives the computer that which makes it more similar to humans: The ability to learn. 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.

The term Machine Learning was coined by Arthur Samuel in 1959, an American pioneer in the field of computer gaming and artificial intelligence and stated that "it gives computers the ability to learn without being explicitly programmed".

And in 1997, Tom Mitchell gave a "well-posed" mathematical and relational definition that "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Machine learning implementations are classified into three major categories, depending on the nature of the learning "signal" or "response" available to a learning system which are :

Supervised learning: When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples. Classification and Regression comes under supervised learning.

Unsupervised learning: Whereas when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of un-correlated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms. Clustering comes under unsupervised learning.

Reinforcement learning: When you present the algorithm with examples that lack labels, as in unsupervised learning. However, you can accompany an example with positive or negative feedback according to the solution the algorithm proposes.

Our target is to classify the ozone level depending on the various factors like temperature at particular height, wind speed etc. given in the dataset. As we have to do classification task, we use support vector machine, random forest classifier and k- nearest neighbour classifier as models to train and predict on the dataset. We compare their performance (specifically accuracy) to check which model works best for our dataset.

PROPOSED METHOD/SOLUTION APPROACH:

Pre-Processing the data:

We did attribute selection where we only kept numeric attributes and deleted the attribute with date datatype from the eighthr.data.csv file (manually). Also we deleted the first row of the 'eighthr.data.csv' (manually) as the attribute names were given in a separate csv file. Later changed it with the names of the attributes from the the csv file 'eighthr.names.csv' (using code).

We changed the datatype of all the columns into float datatype from string datatype (using code) so that we can work on it with numerical functions.

While exploring the dataset, we found that the dataset contains missing values. So we use the interpolation method to fill in the missing values efficiently (using code).

Splitting the data:

We first split our dataset into two parts, input/features and output/label/class. Now we use the train test split method from scikit learn to split our input and output both into train and test dataset.

Training & Testing the model:

We import the models from scikit learn and use their fit method to train on the training data(input and output) and then use the predict method to predict the label on the testing data(input). Also, we tune the hyperparameters of these model to see which model gives the best accuracy.

We used the following models for classification:

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.

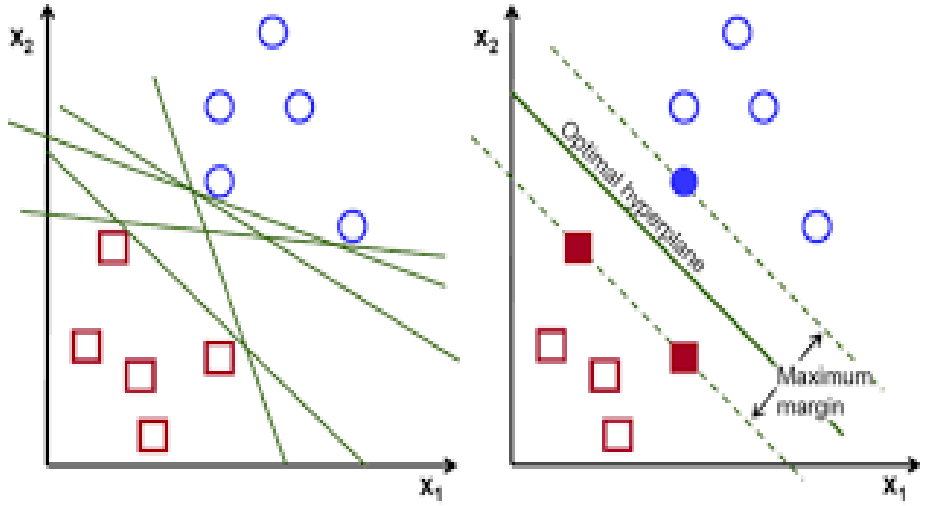
To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function k(x, y) selected to suit the problem.

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.



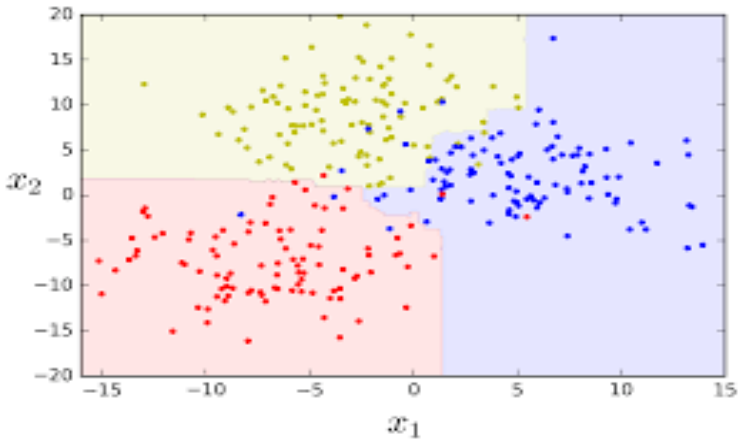
RANDOM FOREST CLASSIFIER:

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).

Distance functions

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

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

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

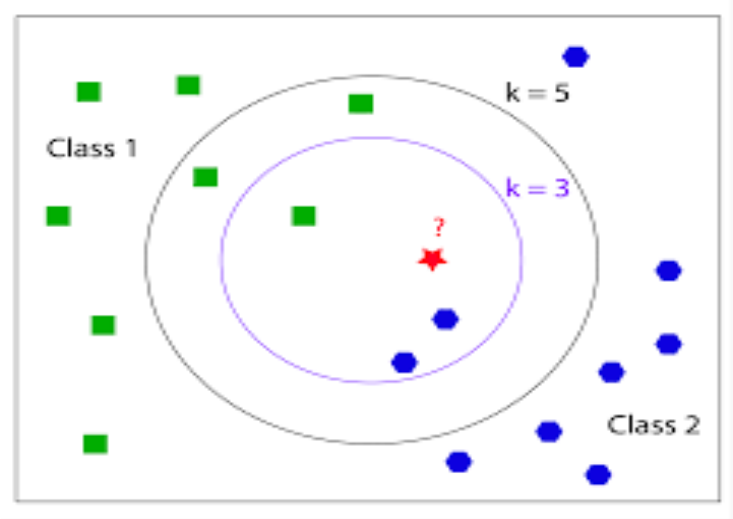
In the context of gene expression microarray data, for example, *k*-NN has also been employed with correlation coefficients such as Pearson and Spearman.

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.



RESULTS & DISCUSSION:

After successful completion of this project we can conclude that Ozone Level Detection Dataset can be best classified by the Random forest classifier.

Here we have used Support Vector Machine, K-Nearest Neighbour, Random Forest 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 0.9358974358974359

In case of Random forest we got an accuracy of 0.9497041420118343

SOURCE CODE

PREPROCESSING THE DATASET

```
#import required modules
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score as accuracy
import matplotlib.pyplot as plt

#reading the dataset header names
dataheader=pd.read_csv("eighthr.names.csv",header=None)
dataheader=dataheader.iloc[1:,0]
dataheader



#creating editted the dataset header list
dh=[]
for i in dataheader:
    s=""
    #print(i)
    for j in i:

        if j!=':':
           s=s+j
        else:
            break
```

```
    dh.append(s)
dh.append("class")
dh=pd.Series(dh)
dh

#creating the dataset
dataset=pd.read_csv("eighthr.data.csv",header=None,names=dh)
ds=dataset.replace("?",np.NaN)
ds

#converting the values in cells from string to float
for i in range(72):
    ds.iloc[:,i]=ds.iloc[:,i].astype(float)

#checking if there exists any missing values
miss=ds.isnull().sum()/len(ds)
miss = miss[miss > 0]
miss.sort_values(inplace=True)
miss

#filling the NaN values by using interpolating process
ds.interpolate(inplace=True)

#checking if there exists any missing values
miss=ds.isnull().sum()/len(ds)
miss = miss[miss > 0]
miss.sort_values(inplace=True)
miss
#checking the final dataset
ds
```

CREATING TRAINING AND TESTING DATASETS

```
#separating dataset into features and lables

x=ds.iloc[:,0:72]

y=ds.iloc[:,72]


#checking the input/features data

x


#checking the output/label data

y


#spliting x and y into training set and testing set

x_train, x_test, y_train, y_test = train_test_split(x,
y,test_size=0.4,random_state=50)


#creating a list to store the best accuracy of the models

best=[]
```

K NEAREST NEIGHBOUR CLASSIFIER

```
#importing module to use knn classifier
from sklearn.neighbors import KNeighborsClassifier

#train knn classifier on training dataset
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)

#predict the output on testing data and check the accuracy
y_predict=knn.predict(x_test)
print(accuracy(y_test,y_predict))

#changing hyperparameter to see which value gives the best result
score1=[]
for k in range(1,100):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_predict=knn.predict(x_test)
    score1.append(accuracy(y_test,y_predict))
print(score1)

plt.plot(score1)

#checking which value of hyperparameter gave the best result
print("k =",score1.index(max(score1)+1),"gives best accuracy
of",max(score1))

#taking the accuracy of knn classifier of comparison
best.append(max(score1))
```

RANDOM FOREST CLASSIFIER

```
#importing module to use random forest classifier
from sklearn.ensemble import RandomForestClassifier

#train random forest classifier on training dataset
rfc=RandomForestClassifier(n_estimators=4)
rfc.fit(x_train,y_train)

#predict the output on testing data and check the accuracy
y_predict=rfc.predict(x_test)
print(accuracy(y_test,y_predict))

#changing hyperparameter to see which value gives the best result
score2=[]
for k in range(1,100):
    rfc = RandomForestClassifier(n_estimators=k)
    rfc.fit(x_train, y_train)
    y_predict=rfc.predict(x_test)
    score2.append(accuracy(y_test,y_predict))
print(score2)
plt.plot(score2)

#checking which value of hyperparameter gave the best result
print("k =",score2.index(max(score2)+1),"gives best accuracy
of",max(score2))

#taking the accuracy of random forest classifier of comparison
best.append(max(score2))
```

SUPPORT VECTOR MACHINE CLASSIFIER

```
#importing module to use support vector machine classifier
from sklearn import svm

#train support vector machine classifier on training dataset
svmc=svm.SVC(gamma='auto')
svmc.fit(x_train,y_train)

#predict the output on testing data and check the accuracy
y_predict=svmc.predict(x_test)
print("gives best accuracy score of",accuracy(y_test,y_predict))

#taking the accuracy of random forest classifier of comparison
best.append(accuracy(y_test,y_predict))
```

THE BEST CLASSIFIER MODEL FOR THIS DATASET

```
#finding the max out of the ‘best’ list to find out who performed
the best

x=best.index(max(best))

if x==0:

    print("K NEAREST NEIGHBOUR IS THE BEST CLASSIFIER")

elif x==1:

    print("RANDOM FOREST IS THE BEST CLASSIFIER")

else:

    print("SUPPORT VECTOR MACHINE IS THE BEST CLASSIFIER")
```

SCREENSHOTS OF OUTPUT

Out[2]:
```
1     WSR0:     continuous.
2     WSR1:     continuous.
3     WSR2:     continuous.
4     WSR3:     continuous.
5     WSR4:     continuous.
                ...
68      KI:     continuous.
69      TT:     continuous.
70     SLP:     continuous.
71    SLP_:     continuous.
72    Precp:    continuous.
Name: 1, Length: 72, dtype: object
```

Out[3]:
```
0      WSR0
1      WSR1
2      WSR2
3      WSR3
4      WSR4
       ...
68       TT
69      SLP
70     SLP_
71    Precp
72    class
Length: 73, dtype: object
```

Out[4]:

	WSR0	WSR1	WSR2	WSR3	WSR4	WSR5	WSR6	WSR7	WSR8	WSR9	...	RH50	U50	V50	HT50	KI	TT	SLP	SLP_	Precp	class
0	0.8	1.8	2.4	2.1	2	2.1	1.5	1.7	1.9	2.3	...	0.15	10.67	-1.56	5795	-12.1	17.9	10330	-55	0	0
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	0.48	8.39	3.84	5805	14.05	29	10275	-55	0	0
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	0.6	6.94	9.8	5790	17.9	41.3	10235	-40	0	0
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	0.49	8.73	10.54	5775	31.15	51.7	10195	-40	2.08	0
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.58	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2529	0.3	0.4	0.5	0.5	0.2	0.3	0.4	0.4	1.3	2.2	...	0.07	7.93	-4.41	5800	-25.6	21.8	10295	65	0	0
2530	1	1.4	1.1	1.7	1.5	1.7	1.8	1.5	2.1	2.4	...	0.04	5.95	-1.14	5845	-19.4	19.1	10310	15	0	0
2531	0.8	0.8	1.2	0.9	0.4	0.6	0.8	1.1	1.5	1.5	...	0.06	7.8	-0.64	5845	-9.6	35.2	10275	-35	0	0
2532	1.3	0.9	1.5	1.2	1.6	1.8	1.1	1	1.9	2	...	0.25	7.72	-0.89	5845	-19.6	34.2	10245	-30	0.05	0
2533	1.5	1.3	1.8	1.4	1.2	1.7	1.6	1.4	1.6	3	...	0.54	13.07	9.15	5820	1.95	39.35	10220	-25	0	0

2534 rows × 73 columns

Out[6]:

```
Precp    0.000789
SLP      0.037490
HT85     0.037490
T85      0.039069
HT70     0.039463
           ...
WSR2     0.116022
WSR20    0.116022
WSR23    0.117206
WSR0     0.117995
WSR22    0.118390
Length: 72, dtype: float64
```

Out[8]:

```
Series([], dtype: float64)
```

Out[10]:

	WSR0	WSR1	WSR2	WSR3	WSR4	WSR5	WSR6	WSR7	WSR8	WSR9	...	RH50	U50	V50	HT50	KI	TT	SLP	SLP_	Precp
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	0.15	10.670	-1.56	5795.0	-12.10	17.900	10330.0	-55.000000	0.00
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	0.48	8.390	3.84	5805.0	14.05	29.000	10275.0	-55.000000	0.00
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	0.60	6.940	9.80	5790.0	17.90	41.300	10235.0	-40.000000	0.00
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	0.49	8.730	10.54	5775.0	31.15	51.700	10195.0	-40.000000	2.08
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	0.29	10.355	10.91	5772.5	29.55	48.975	10157.5	-53.333333	0.58
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2529	0.3	0.4	0.5	0.5	0.2	0.3	0.4	0.4	1.3	2.2	...	0.07	7.930	-4.41	5800.0	-25.60	21.800	10295.0	65.000000	0.00
2530	1.0	1.4	1.1	1.7	1.5	1.7	1.8	1.5	2.1	2.4	...	0.04	5.950	-1.14	5845.0	-19.40	19.100	10310.0	15.000000	0.00
2531	0.8	0.8	1.2	0.9	0.4	0.6	0.8	1.1	1.5	1.5	...	0.06	7.800	-0.64	5845.0	-9.60	35.200	10275.0	-35.000000	0.00
2532	1.3	0.9	1.5	1.2	1.6	1.8	1.1	1.0	1.9	2.0	...	0.25	7.720	-0.89	5845.0	-19.60	34.200	10245.0	-30.000000	0.05
2533	1.5	1.3	1.8	1.4	1.2	1.7	1.6	1.4	1.6	3.0	...	0.54	13.070	9.15	5820.0	1.95	39.350	10220.0	-25.000000	0.00

2534 rows × 73 columns

Out[12]:

	WSR0	WSR1	WSR2	WSR3	WSR4	WSR5	WSR6	WSR7	WSR8	WSR9	...	T50	RH50	U50	V50	HT50	KI	TT	SLP	SLP_
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	-15.5	0.15	10.670	-1.56	5795.0	-12.10	17.900	10330.0	-55.000000
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	-14.5	0.48	8.390	3.84	5805.0	14.05	29.000	10275.0	-55.000000
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	-15.9	0.60	6.940	9.80	5790.0	17.90	41.300	10235.0	-40.000000
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	-16.8	0.49	8.730	10.54	5775.0	31.15	51.700	10195.0	-40.000000
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	-14.3	0.29	10.355	10.91	5772.5	29.55	48.975	10157.5	-53.333333
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2529	0.3	0.4	0.5	0.5	0.2	0.3	0.4	0.4	1.3	2.2	...	-12.4	0.07	7.930	-4.41	5800.0	-25.60	21.800	10295.0	65.000000
2530	1.0	1.4	1.1	1.7	1.5	1.7	1.8	1.5	2.1	2.4	...	-12.0	0.04	5.950	-1.14	5845.0	-19.40	19.100	10310.0	15.000000
2531	0.8	0.8	1.2	0.9	0.4	0.6	0.8	1.1	1.5	1.5	...	-11.8	0.06	7.800	-0.64	5845.0	-9.60	35.200	10275.0	-35.000000
2532	1.3	0.9	1.5	1.2	1.6	1.8	1.1	1.0	1.9	2.0	...	-10.8	0.25	7.720	-0.89	5845.0	-19.60	34.200	10245.0	-30.000000
2533	1.5	1.3	1.8	1.4	1.2	1.7	1.6	1.4	1.6	3.0	...	-11.9	0.54	13.070	9.15	5820.0	1.95	39.350	10220.0	-25.000000

2534 rows × 72 columns

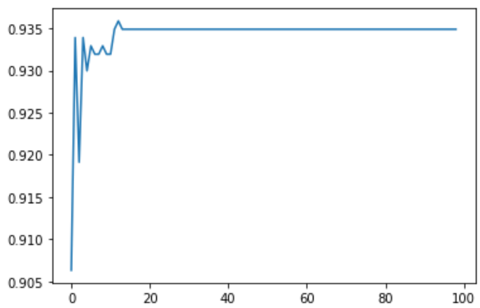
```
Out[13]: 0       0
         1       0
         2       0
         3       0
         4       0
                ..
         2529    0
         2530    0
         2531    0
         2532    0
         2533    0
         Name: class, Length: 2534, dtype: int64
```

```
Out[17]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                              weights='uniform')
```

```
0.9299802761341223
```

```
[0.9063116370808678, 0.9339250493096647, 0.9191321499013807, 0.9339250493096647, 0.9299802761341223, 0.9329388560157791, 0.9319
526627218935, 0.9319526627218935, 0.9329388560157791, 0.9319526627218935, 0.9319526627218935, 0.9349112426035503, 0.93589743589
74359, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503,
0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.93491
12426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.934911242603
5503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503,
0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.93491
12426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.934911242603
5503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503,
0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.93491
12426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.934911242603
5503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503,
0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.93491
12426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.934911242603
5503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503,
0.9349112426035503, 0.9349112426035503, 0.9349112426035503, 0.9349112426035503]
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x1e46d99d848>]
```



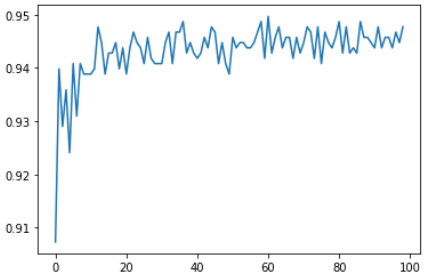

```
k = 13 gives best accuracy of 0.9358974358974359
```

```
Out[23]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                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)
```

0.9358974358974359

```
[0.9072978303747534, 0.9398422090729783, 0.9289940828402367, 0.9358974358974359, 0.9240631163708086, 0.9408284023668639, 0.9309
664694280079, 0.9408284023668639, 0.9388560157790927, 0.9388560157790927, 0.9388560157790927, 0.9398422090729783, 0.94773175542
40631, 0.9447731755424064, 0.9388560157790927, 0.9428007889546351, 0.9428007889546351, 0.9447731755424064, 0.9398422090729783,
0.9437869822485208, 0.9388560157790927, 0.9437869822485208, 0.9467455621301775, 0.9447731755424064, 0.9437869822485208, 0.94082
84023668639, 0.9457593688362919, 0.9418145956607495, 0.9408284023668639, 0.9408284023668639, 0.9408284023668639, 0.944773175542
4064, 0.9467455621301775, 0.9408284023668639, 0.9467455621301775, 0.9467455621301775, 0.9487179487179487, 0.9428007889546351,
0.9447731755424064, 0.9428007889546351, 0.9418145956607495, 0.9428007889546351, 0.9457593688362919, 0.9437869822485208, 0.94773
17554240631, 0.9467455621301775, 0.9408284023668639, 0.9447731755424064, 0.9408284023668639, 0.9388560157790927, 0.945759368836
2919, 0.9437869822485208, 0.9447731755424064, 0.9447731755424064, 0.9437869822485208, 0.9437869822485208, 0.9447731755424064,
0.9467455621301775, 0.9487179487179487, 0.9418145956607495, 0.9497041420118343, 0.9428007889546351, 0.9457593688362919, 0.94773
17554240631, 0.9437869822485208, 0.9457593688362919, 0.9457593688362919, 0.9418145956607495, 0.9457593688362919, 0.942800788954
6351, 0.9447731755424064, 0.9477317554240631, 0.9467455621301775, 0.9418145956607495, 0.9477317554240631, 0.9408284023668639,
0.9467455621301775, 0.9447731755424064, 0.9437869822485208, 0.9457593688362919, 0.9487179487179487, 0.9428007889546351, 0.94773
17554240631, 0.9428007889546351, 0.9437869822485208, 0.9428007889546351, 0.9487179487179487, 0.9457593688362919, 0.945759368836
2919, 0.9447731755424064, 0.9437869822485208, 0.9477317554240631, 0.9437869822485208, 0.9457593688362919, 0.9457593688362919,
0.9437869822485208, 0.9467455621301775, 0.9447731755424064, 0.9477317554240631]
```

Out[25]: [<matplotlib.lines.Line2D at 0x1e46db70f48>]



k = 61 gives best accuracy of 0.9497041420118343

Out[29]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

gives best accuracy score of 0.9349112426035503

RANDOM FOREST IS THE BEST CLASSIFIER