

Practical No 1

Date:

Aim: Installation and working on various AI tools like python, R tool, GATE, etc.

Theory:

- **Introduction:**

In today's data-driven world, AI tools play a crucial role in extracting insights and making informed decisions. This experiment focuses on three popular AI tools: Python, R, and GATE. Python is a versatile programming language known for its simplicity and extensive libraries. R is a statistical programming language with a wide range of packages for statistical analysis and machine learning. GATE is an open-source software framework specifically designed for natural language processing tasks.

- **Python:**

Python is a versatile programming language widely used in the AI community due to its simplicity, readability, and extensive libraries. It offers various AI-specific packages such as NumPy, Pandas, TensorFlow, Keras, and scikit-learn. These packages enable data manipulation, scientific computing, machine learning, and deep learning tasks. Python serves as the backbone for AI development, allowing researchers to implement algorithms and build models effectively.

- **R Tool:**

R is a statistical programming language known for its comprehensive collection of packages and libraries specifically designed for statistical analysis, data visualization, and machine learning. It is popular among statisticians and data scientists for its robustness in handling complex data and conducting advanced statistical modeling. R provides a vast ecosystem of packages like caret, randomForest, ggplot2, and dplyr, which empower researchers to perform sophisticated data analysis and modeling tasks.

- **GATE (General Architecture for Text Engineering):**

GATE is an open-source software framework and development environment primarily used for natural language processing (NLP) tasks. It enables researchers to develop and deploy NLP pipelines for tasks such as information extraction, text mining, sentiment analysis, and machine translation. GATE offers a rich set of tools, resources, and pre-trained models for NLP, making it a valuable tool for AI practitioners working with textual data.

- **Installation and Setup:**

To conduct the experiment, the first step involves downloading and installing the required software. For Python, researchers can download and install the latest version of Python from the official website or utilize Python distributions like Anaconda or Miniconda. For R, the latest version of R can be downloaded from the Comprehensive R Archive Network (CRAN). GATE can be downloaded and installed from the official GATE website

Step 1 – Select Version of Python to Install

Python has various versions available with differences between the syntax and working of different versions of the language. We need to choose the version which we want to use or need. There are different versions of Python 2 and Python 3 available.

Step 2 – Download Python Executable Installer

On the web browser, in the official site of python (www.python.org), move to the Download for Windows section.

All the available versions of Python will be listed. Select the version required by you and click on Download. Let suppose, we chose the Python 3.10.4 version.

Python Releases for Windows

- [Latest Python 3 Release - Python 3.10.4](#)
- [Latest Python 2 Release - Python 2.7.18](#)

Stable Releases

- [Python 3.10.4 - March 24, 2022](#)

Note that Python 3.10.4 cannot be used on Windows 7 or earlier.

- [Download Windows embeddable package \(32-bit\)](#)
- [Download Windows embeddable package \(64-bit\)](#)
- [Download Windows help file](#)
- [Download Windows installer \(32-bit\)](#)
- [Download Windows installer \(64-bit\)](#)

On clicking download, various available executable installers shall be visible with different operating system specifications. Choose the installer which suits your system operating system and download the installer. Let suppose, we select the Windows installer(64 bits). The download size is less than 30MB.

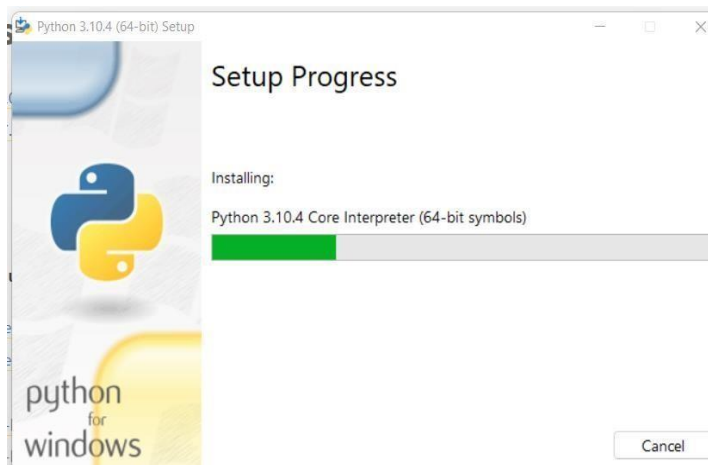
Step 3 – Run Executable Installer

We downloaded the Python 3.9.1 Windows 64 bit installer.

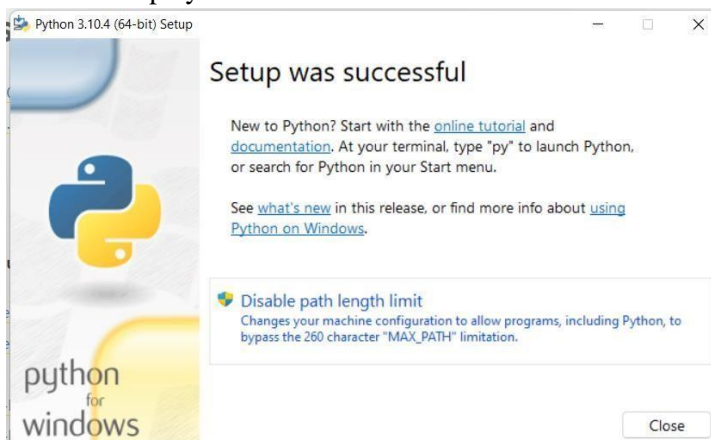
Run the installer. Make sure to select both the checkboxes at the bottom and then click Install New.



On clicking the Install Now, The installation process starts.

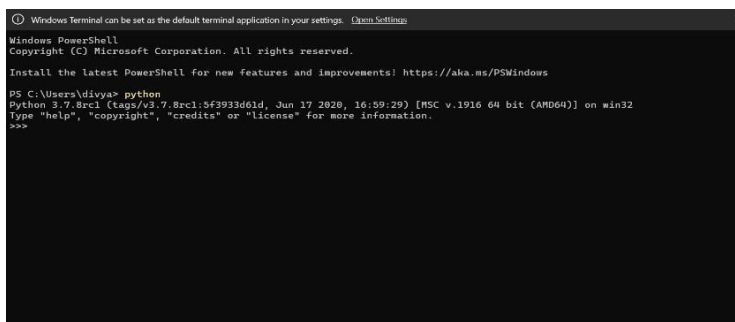


The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.



Step 4 – Verify Python is installed on Windows

To ensure if Python is successfully installed on your system. Follow the given steps – Open the command prompt. Type 'python' and press enter.



```
Windows Terminal can be set as the default terminal application in your settings. Learn Settings
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\divya> python
Python 3.7.8rc1 (tags/v3.7.8rc1:5f393d61d, Jun 17 2020, 16:59:29) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

The version of the python which you have installed will be displayed if the python is successfully installed on your windows.

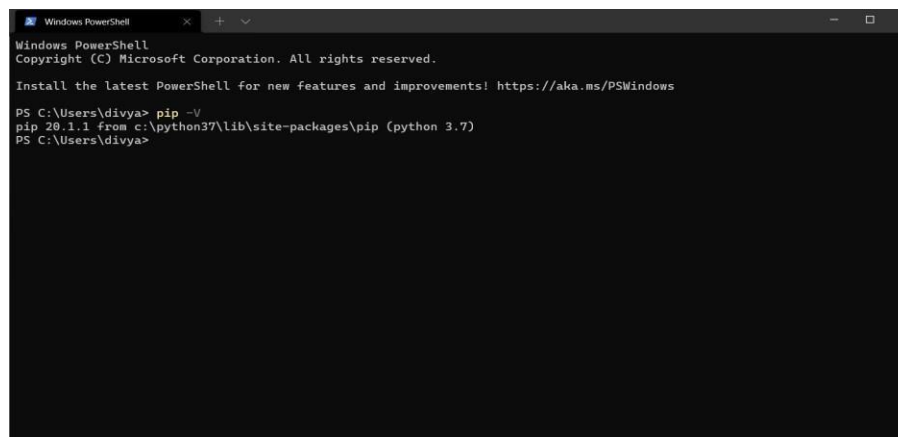
Step 5 – Verify Pip was installed

Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

To verify if pip was installed, follow the given steps – Open the command prompt.

Enter `pip -V` to check if pip was installed.

The following output appears if pip is installed successfully.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\divya> pip -V
pip 20.1.1 from c:\python37\lib\site-packages\pip (python 3.7)
PS C:\Users\divya>
```

We have successfully installed python and pip on our Windows system.

Python also features dynamic type system and automatic memory management supporting a wide variety of programming paradigms including object-oriented, imperative, functional and procedural to name a few.

Python is available for all Operating Systems and also has an open-source offering titled CPython which is garnering widespread popularity as well.

- **Conclusion:**

Through this experiment, we gained practical experience in installing and working with essential AI tools like Python, R, and GATE. This hands-on exploration helped us understand the functionalities and capabilities of each tool, enabling us to make informed decisions about their usage in future projects. By analyzing the results and comparing the performance of these tools, we gained valuable insights into their

strengths and limitations. Overall, this experiment enhanced our understanding of AI tools and their applications in real-world scenarios, providing a solid foundation for our future endeavors in AI and data analysis.

Practical No 2

Date:

Aim: Implementation of Data balancing in Python using appropriate database.

Theory:

1) Imbalanced Dataset:

An imbalanced dataset is characterized by a significant difference in the number of samples between different classes. In the cerebral stroke dataset, the target column contains binary values (0 and 1), where 0 represents no stroke and 1 represents a stroke. If the distribution of these values is highly imbalanced, the dataset needs to be balanced to avoid bias in the model's predictions.

2) Balancing Techniques:

There are several techniques available to balance an imbalanced dataset, including oversampling and undersampling:

3) Oversampling: Oversampling involves increasing the number of samples in the minority class to match the number of samples in the majority class. This can be achieved by replicating existing minority samples or generating synthetic samples using techniques such as SMOTE (Synthetic Minority Over-sampling Technique).

4) Undersampling: Undersampling involves reducing the number of samples in the majority class to match the number of samples in the minority class. This can be done by randomly selecting a subset of the majority class samples.

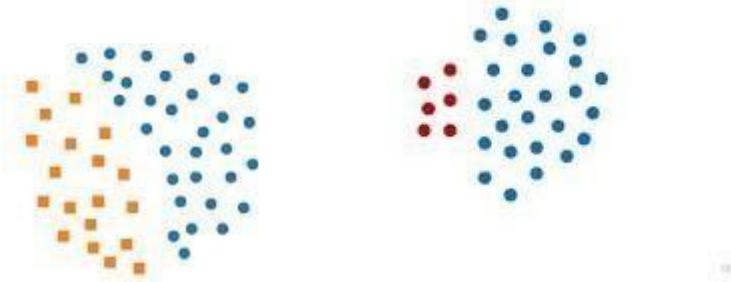
5) Resampling with Replacement:

Resampling with replacement is a technique commonly used in oversampling. It involves randomly selecting samples from the minority class with replacement, allowing the same sample to be selected multiple times. This helps in increasing the number of minority class samples to balance the dataset.

6) Combining Balanced Classes:

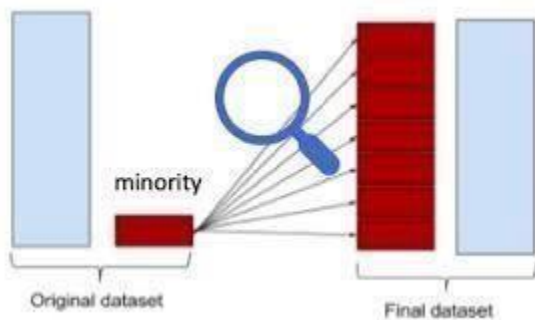
After oversampling or undersampling, the balanced dataset is created by combining the upsampled or downsampled minority class with the majority class. This ensures that the number of samples in each class is approximately equal.

Imbalanced Class Distribution



Techniques to Convert Imbalanced Dataset into Balanced Dataset

Imbalanced data is not always a bad thing, and in real data sets, there is always some degree of imbalance. That said, there should not be any big impact on your model performance if the level of imbalance is relatively low.



Now, let's cover a few techniques to solve the class imbalance problem.

1 — Use the right evaluation metrics:

Evaluation metrics can be applied such as:

Confusion Matrix: a table showing correct predictions and types of incorrect predictions. **Precision:** the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.

Recall: the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.

F1-Score: the weighted average of precision and recall.

2 — Over-sampling (Up Sampling):

This technique is used to modify the unequal data classes to create balanced datasets. When the quantity of data is insufficient, the oversampling method tries to balance by incrementing the size of rare samples. (Or)

Over-sampling increases the number of minority class members in the training set. The advantage of over-sampling is that no information from the original training

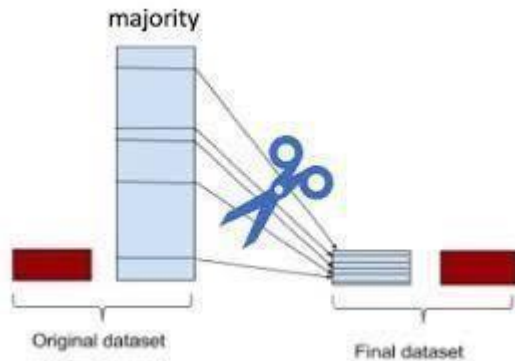
Advantages

No loss of information

Mitigate overfitting caused by oversampling. Disadvantages Overfitting

3 — Under-sampling (Down Sampling): Unlike oversampling, this technique balances the imbalance dataset by reducing the size of the class which is in abundance. There are various methods for classification problems such as cluster centroids and Tomek links. and the Tomek link method removes unwanted overlap between classes until all minimally distanced nearest neighbors are of the same class. (Or)

Under-sampling, on contrary to over-sampling, aims to reduce the number of majority samples to balance the class distribution. Since it is removing observations from the original data set, it might discard useful information.



Advantages

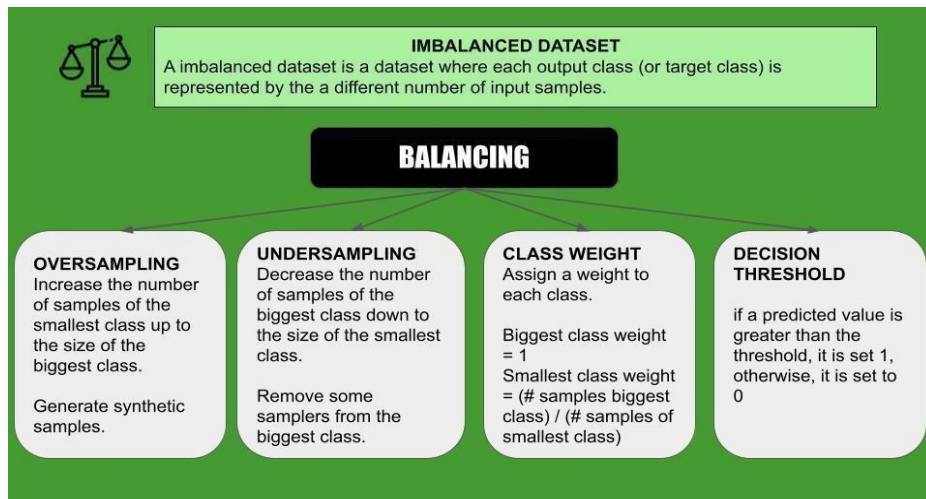
Run-time can be improved by decreasing the amount of training dataset.

Helps in solving the memory problems Disadvantages

Losing some critical information

4 — Feature selection: In order to tackle the imbalance problem, we calculate the one-sided metric such as correlation coefficient (CC) and odds ratios (OR) or twosided metric evaluationsuch as information gain (IG) and chi-square (CHI) on both the positive class and negative class.

Identifying these features will help us generate a clear decision boundary with respect to each class. This helps the models o classify the data more accurately. This performs the function of intelligent subsampling and potentially helps reduce the imbalance problem.



NEED FOR DATA BALANCING:

Correcting Previous Mistakes from Imbalanced Datasets:

- Never test on the oversampled or under sampled dataset.
- If we want to implement cross validation, remember to oversample or under sample your training data during cross-validation, not before!
- Don't use accuracy score as a metric with imbalanced datasets (will be usually high and misleading), instead use f1-score, precision/recall score or confusion matrix

EXECUTION:-

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets pre
# You can also write temporary files to /kaggle/temp/, but they won't be saved outsi
```

In [9]:

```
df = pd.read_csv('/kaggle/input/creditcardcsv/creditcard.csv')
```

In [9]:

```
df = pd.read_csv('/kaggle/input/creditcardcsv/creditcard.csv')
```

In [24]:

```
x = df.drop(['Class'],axis=1)
y = df['Class']
y.value_counts()
```

Out[24]:

```
0    284315
1       492
Name: Class, dtype: int64
```

In [16]:

```
x.columns
```

Out[16]:

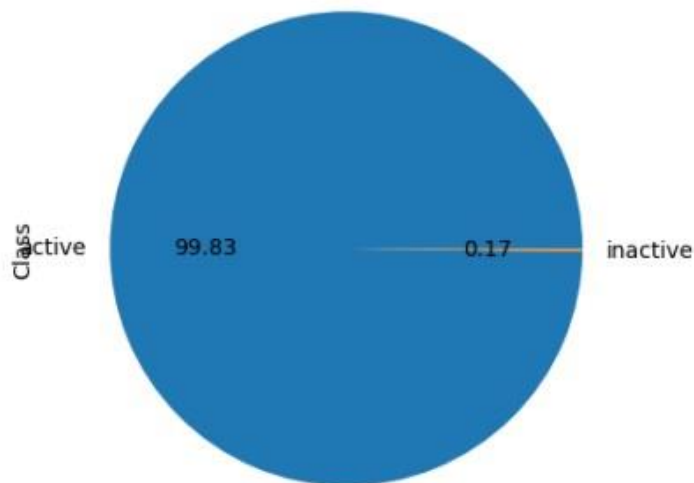
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9',
      'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',
      'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'],
      dtype='object')
```

In [18]:

```
y.value_counts().plot.pie(autopct='%0.2f',labels=['active','inactive'])
```

Out[18]:

<AxesSubplot: ylabel='Class'>

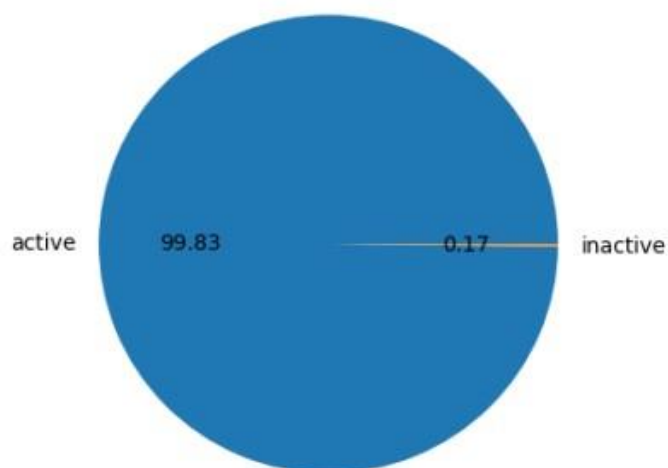


In [23]:

```
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
ax1.pie(y.value_counts(), autopct='%.2f', labels=['active', 'inactive'])
```

Out[23]:

```
(<matplotlib.patches.Wedge at 0x7637ef856530>,
 <matplotlib.patches.Wedge at 0x7637ef856200>],
 [Text(-1.09998380137016, 0.0059696501784341355, 'active'),
 Text(1.0999838018177286, -0.005969567707642625, 'inactive')],
 [Text(-0.5999911643837235, 0.0032561728246004373, '99.83'),
 Text(0.5999911646278518, -0.0032561278405323405, '0.17')])
```



In [32]:

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy=1)
X_res, y_res = rus.fit_resample(X,y)

ax = y_res.value_counts().plot.pie(autopct='%.2f', labels=['active', 'inactive'])
ax.set_title("Under-sampling")
```

Out[32]:

```
Text(0.5, 1.0, 'Under-sampling')
```

In [28]:

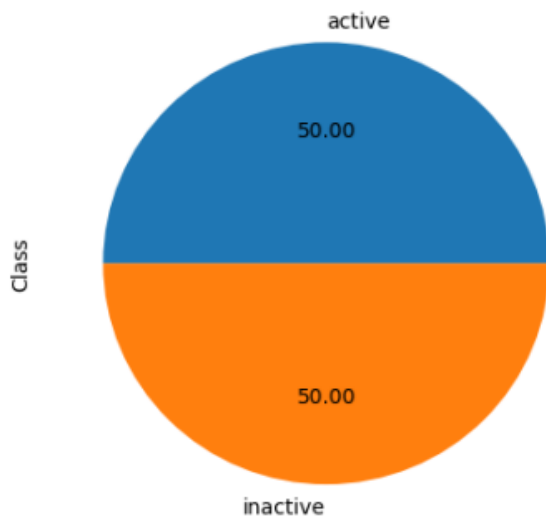
```
x_res
```

Out[28]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	38287.0	-0.325757	0.185395	2.492519	1.160467	-0.761722	0.617422	0.022260	0.151
1	38363.0	0.425038	-0.833703	0.531609	2.555723	-0.521557	0.788768	-0.013621	0.262
2	149159.0	1.810875	0.533045	-2.196606	3.720310	1.367523	-0.500415	1.251329	-0.454
3	145123.0	0.308120	0.850066	0.300567	1.026995	0.402899	-0.601644	0.770920	-0.459
4	68404.0	-0.396829	0.970653	1.330348	-0.137273	0.095416	-0.602696	0.708046	-0.013
...
979	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697
980	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248
981	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210
982	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058
983	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068

984 rows x 30 columns

Under-sampling



In [26]:

```
y_res.value_counts()
```

Out[26]:

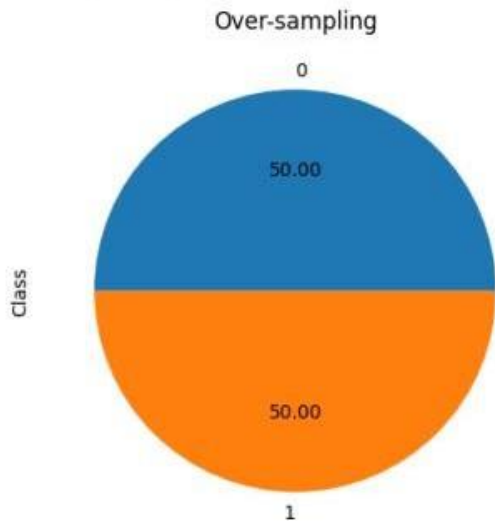
```
0    492
1    492
```

Name: Class, dtype: int64

In [29]:

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(sampling_strategy="not majority") # String
X_res, y_res = ros.fit_resample(X, y)

ax = y_res.value_counts().plot.pie(autopct='%0.2f')
_ = ax.set_title("Over-sampling")
```



In [30]:

```
y_res.value_counts()
```

Out[30]:

```
0    284315
1    284315
Name: Class, dtype: int64
```

In [31]:

```
X_res
```

Out[31]:

In [31]:

X_res

Out[31]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
...
568625	93853.0	-6.185857	7.102985	-13.030455	8.010823	-7.885237	-3.974550	-12.229608
568626	34687.0	-0.860827	3.131790	-5.052968	5.420941	-2.494141	-1.811287	-5.479117
568627	139816.0	-0.395582	-0.751792	-1.984666	-0.203459	1.903967	-1.430289	-0.076548
568628	93860.0	-10.850282	6.727466	-16.760583	8.425832	-10.252697	-4.192171	-14.077086
568629	84204.0	-0.937843	3.462889	-6.445104	4.932199	-2.233983	-2.291561	-5.695594

568630 rows x 30 columns

CONCLUSION:

Balancing an imbalanced cerebral stroke dataset is crucial to prevent biased predictions and ensure that the machine learning model can learn from all classes effectively. By using techniques such as oversampling or undersampling, it is possible to create a balanced dataset where the number of samples in each class is approximately equal. The balanced dataset can then be used to train and evaluate machine learning models to make accurate predictions for stroke occurrences.

Practical No 3

Date:

Aim: Perform Outlier detection in Python using appropriate libraries and tools on appropriate dataset

Introduction:

Outlier detection is a critical task in data analysis and plays a significant role in identifying unusual or abnormal observations within a dataset. Outliers can be caused by various factors such as measurement errors, data corruption, or genuinely rare events. Detecting and handling outliers is essential to ensure the integrity and accuracy of data analysis results. The Z-score method is a statistical approach commonly used for outlier detection. This theory discusses the Z-score method and its application in identifying outliers in a dataset.

Theory:

1. Z-Score:

- The Z-score, also known as the standard score, measures how many standard deviations a data point is away from the mean of a distribution.
- It is calculated by subtracting the mean from the data point and dividing the result by the standard deviation.
- The formula for calculating the Z-score is: $Z = (X - \mu) / \sigma$, where X is the data point, μ is the mean, and σ is the standard deviation.

2. Z-Score Method for Outlier Detection:

- The Z-score method is based on the assumption that data points in a distribution follow a Gaussian (normal) distribution.
- Outliers are considered observations that deviate significantly from the mean of the distribution.
- In the Z-score method, a threshold value is set to determine when a data point is considered an outlier.
- Typically, a threshold value of 2 or 3 is used, where data points with a Z-score greater than the threshold are considered outliers.

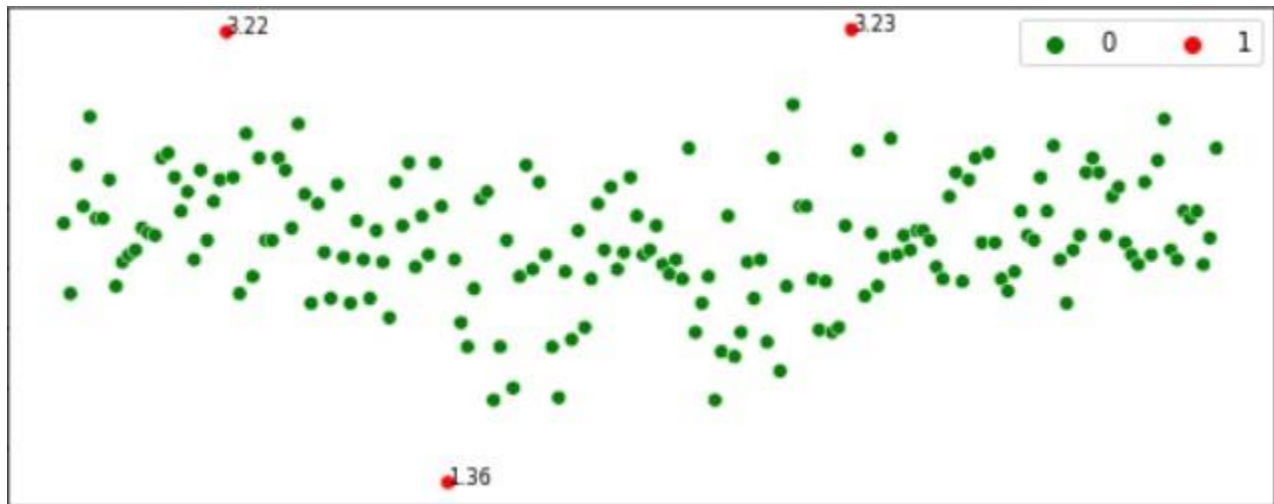
3. Steps for Outlier Detection using Z-Score:

- Load the dataset into a suitable data structure (e.g., DataFrame).
- Calculate the Z-scores for the variables of interest (e.g., height, weight).
- Subtract the mean of the variable from each data point and divide it by the standard deviation.
- Set a threshold value for outlier detection (e.g., Z-score > 3).
- Identify and flag the data points that exceed the threshold value as outliers.
- Analyze and interpret the outliers in the context of the specific dataset and problem domain.

4. Interpretation and Handling of Outliers:

- Once the outliers have been identified, it is crucial to interpret their significance.
- Outliers may indicate data measurement errors, data entry mistakes, or genuine anomalies in the data.
- Careful consideration is required to determine whether outliers should be removed, corrected, or

What are outliers?



Outliers are the values that look different from the other values in the data. Below is a plot highlighting the outliers in 'red' and outliers can be seen in both the extremes of data.

Reasons for outliers in data

1. Errors during data entry or a faulty measuring device (a faulty sensor may result in extreme readings).
2. Natural occurrence (salaries of junior level employees vs C-level employees)

Problems caused by outliers

1. Outliers in the data may causes problems during model fitting (esp. linear models).
2. Outliers may inflate the error metrics which give higher weights to large errors (example, mean squared error, RMSE).

Methods to identify outliers in the data

We'll use wine dataset of Scikit-Learn. Before proceeding further, we'll load and prepare the data.

1. Box plots

Box plots are a visual method to identify outliers. Box plots is one of the many ways to visualize data distribution. Box plot plots the q_1 (25th percentile), q_2 (50th percentile or median) and q_3 (75th percentile) of the data along with $(q_1 - 1.5 * (q_3 - q_1))$ and $(q_3 + 1.5 * (q_3 - q_1))$. Outliers, if any, are plotted as points above and below the plot.

2. IQR method

IQR method is used by box plot to highlight outliers. IQR stands for interquartile range, which is the difference

between q3 (75th percentile) and q1 (25th percentile). The IQR method computes lower bound and upper bound to identify outliers.

Lower Bound = $q1 - 1.5 * IQR$ Upper Bound = $q3 + 1.5 * IQR$

Any value below the lower bound and above the upper bound are considered to be outliers. Below is the implementation of IQR method in Python.

3. Z-score method

Z-score method is another method for detecting outliers. This method is generally used when a variable's distribution looks close to Gaussian. Z-score is the number of standard deviations a value of a variable is away from the variable's mean.

$Z\text{-Score} = (X - \text{mean}) / \text{Standard deviation}$

when the values of a variable are converted to Z-scores, then the distribution of the variable is called standard normal distribution with mean=0 and standard deviation=1. The Z-score method requires a cut-off specified by the user, to identify outliers. The widely used lower end cut-off is

-3 and the upper end cut-off is +3. The reason behind using these cut-offs is, 99.7% of the values lie between -3 and +3 in a standard normal distribution. Let's look at the implementation of Z-Score method in Python.

4. 'Distance from the mean' method (Multivariate method)

Unlike the previous methods, this method considers multiple variables in a data set to detect outliers. This method calculates the Euclidean distance of the data points from their mean and converts the distances into absolute z-scores. Any z-score greater than the pre-specified cut-off is considered to be an outlier.

We'll consider two variables ('malic_acid' and 'magnesium') of the wine dataset for implementing this method in Python using a cut-off of 3.

Removing the outliers :-

For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used.

SOURCE CODE:

```
from collections import Counter
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("banknote s.csv")
print(df.head())
```

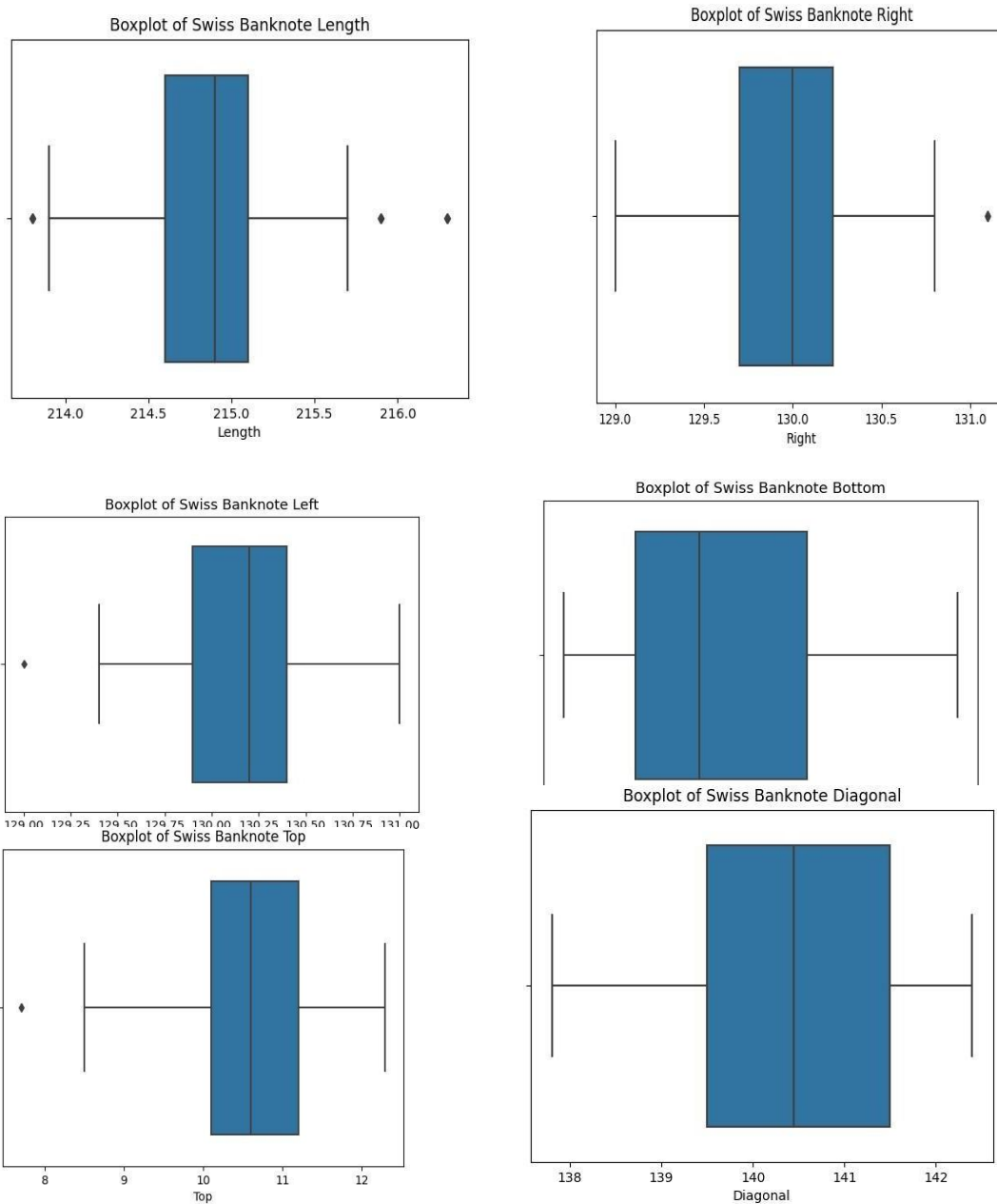
```
sns.boxplot(data=df, x=df["Length"])
plt.title("Boxplot of Swiss Banknote Length ")
df_outlier1 = df[df["Length"] > 216].copy()
```

```
print(Counter(df_outlier1['conterfeit']))
df_outlier2 = df[df["Length"] > 215.5].copy()
print(Counter(df_outlier2['conterfeit']))
def boxplot(column):
```

```
sns.boxplot(data=df, x=df[f"{column}"])
plt.title(f"Boxplot of Swiss Banknote {column}")
```

```
plt.show() boxplot('Length') boxplot('Right') boxplot('Left') boxplot('Bottom') boxplot('Top')
boxplot('Diagonal')
df_outlier3 = df[(df['Length']> 215)&(df['Right']> 130)&(df['Left']> 130)&(df['Bottom']> 10)].copy()
print(Counter(df_outlier3['conterfeit']))
```

Ouput :



Conclusion:

The Z-score method provides a statistical approach to detect outliers in a dataset. By calculating the Z-scores for the variables of interest, we can identify data points that significantly deviate from the mean. Outlier detection

helps ensure data integrity and enables researchers to make informed decisions regarding their treatment. However, it is important to interpret and handle outliers appropriately, considering the context and specific requirements of the analysis. Outlier detection is a valuable tool for data analysis, aiding in identifying and addressing data anomalies effectively.

PRACTICAL 4

AIM: Implement A* Path Finding Algorithm.

THEORY: $F = G + H$

One important aspect of A* is $f = g + h$. The f , g , and h variables are in our Node class and get calculated every time we create a new node. Quickly I'll go over what these variables mean.

- F is the total cost of the node.
- G is the distance between the current node and the start node.
- H is the heuristic — estimated distance from the current node to the end node.

Let's take a look at a quick graphic to help illustrate this.

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

wow such numbers, very color

Awesome! Let's say node(0) is our starting position and node(19) is our end position. Let's also say that our current node is at the red square node(4).

G

G is the distance between the current node and the start node.

If we count back we can see that node(4) is 4 spaces away from our start node.

We can also say that G is 1 more than our parent node (node(3)). So in this case for node(4), $\text{currentNode.g} = 4$.

H

H is the heuristic — estimated distance from the current node to the end node.

So let's calculate the distance. If we take a look we'll see that if we go over 7 spaces and up 3 spaces, we've reached our end node (node(19)).

Let's apply the Pythagorean Theorem! $a^2 + b^2 = c^2$. After we've applied this, we'll see that $\text{currentNode.h} = 7^2 + 3^2$. Or $\text{currentNode.h} = 58$.

But don't we have to apply the square root to 58? Nope! We can skip that calculation on every node and still get the same output. Clever!

With a heuristic, we need to make sure that we can actually calculate it. It's also very important that the heuristic is always an underestimation of the total path, as an overestimation will lead to A* searching for through nodes that may not be the 'best' in terms of f value.

F

F is the total cost of the node.

So let's add up h and g to get the total cost of our node. $\text{currentNode.f} = \text{currentNode.g} + \text{currentNode.h}$. Or $\text{currentNode.f} = 4 + 58$. Or $\text{currentNode.f} = 62$.

Time to use $f = g + h$

Alright, so that was a lot of work. Now with all that work, what am I going to use this f value for?

With this new f value, we can look at all our nodes and say, "Hey, is this the best node I can pick to move forward with right now?". Rather than running through every node, we can pick the ones that have the best chance of getting us to our goal.

Here's a graphic to illustrate. On top, we have Dijkstra's Algorithm, which searches without this f value, and below we have A* which does use this f value.

Dijkstra's Algorithm

So taking a look at Dijkstra's algorithm, we see that it just keeps searching. It has no idea which node is 'best', so it calculates paths for them all.

A* Algorithm

With A*, we see that once we get past the obstacle, the algorithm prioritizes the node with the lowest f and the 'best' chance of reaching the end.

A StarMethod Steps — from Patrick Lester

1. Add the starting square (or node) to the open list.
2. Repeat the following:

A) Look for the lowest F cost square on the open list. We refer to this as the current square.

B). Switch it to the closed list.

C) For each of the 8 squares adjacent to this current square ...

- If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
- If it isn't on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
- If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. If you are keeping your open list sorted by F score, you may need to resort the list to account for the change.

D) Stop when you:

- Add the target square to the closed list, in which case the path has been found, or
- Fail to find the target square, and the open list is empty. In this case, there is no path.

3. Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is your path.

PSUEDOCODE:

// A* Search Algorithm

1. Initialize the open list

2. Initialize the closed list

put the starting node on the open list (you can leave its f at zero)

3. while the open list is not empty

a) find the node with the least f on the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their parents to q

d) for each successor

```
#include <bits/stdc++.h>
```

```
using namespace std; #define ROW 9
```

```
#define COL 10
```

```
// Creating a shortcut for int, int pair type
```

```
typedef pair<int, int> Pair;
```

```

// Creating a shortcut for pair<int, pair<int, int>> type typedef pair<double, pair<int, int> > pPair;
// A structure to hold the necessary parameters struct cell {
// Row and Column index of its parent

// Note that 0 <= i <= ROW-1 & 0 <= j <= COL-1 int parent_i, parent_j;
// f = g + h double f, g, h;
};

bool isValid(int row, int col)

{

// Returns true if row number and column number

// is in range

return (row >= 0) && (row < ROW) && (col >= 0) && (col < COL);
}

// A Utility Function to check whether the given cell is

// blocked or not

bool isUnBlocked(int grid[][COL], int row, int col)

{

// Returns true if the cell is not blocked else false if (grid[row][col] == 1)
return (true);

else

return (false);

}

// A Utility Function to check whether destination cell has

// been reached or not

bool isDestination(int row, int col, Pair dest)

{

if (row == dest.first && col == dest.second) return (true);
else

}

```

```
        return (false);

double calculateHValue(int row, int col, Pair dest)

{

// Return using the distance formula return ((double)sqrt(
(row - dest.first) * (row - dest.first)

+ (col - dest.second) * (col - dest.second)));

}

void tracePath(cell cellDetails[][COL], Pair dest)

{

printf("\nThe Path is "); int row = dest.first;
int col = dest.second;

stack<Pair> Path;
```



```

while (!(cellDetails[row][col].parent_i == row
&& cellDetails[row][col].parent_j == col)) { Path.push(make_pair(row, col));
int temp_row = cellDetails[row][col].parent_i; int temp_col = cellDetails[row][col].parent_j; row = temp_row;
col = temp_col;

}

Path.push(make_pair(row, col)); while (!Path.empty()) {
pair<int, int> p = Path.top(); Path.pop();
printf("-> (%d,%d) ", p.first, p.second);

}

return;

}

void aStarSearch(int grid[][COL], Pair src, Pair dest)

{

// If the source is out of range

if (isValid(src.first, src.second) == false) { printf("Source is invalid\n"); return;
}
if (isValid(dest.first, dest.second) == false) { printf("Destination is invalid\n");

```

```

return;

}

if (isUnBlocked(grid, src.first, src.second) == false

|| isUnBlocked(grid, dest.first, dest.second)

== false) {

printf("Source or the destination is blocked\n"); return;
}

if (isDestination(src.first, src.second, dest)

== true) {

printf("We are already at the destination\n"); return;
}

bool closedList[ROW][COL]; memset(closedList, false, sizeof(closedList)); cell cellDetails[ROW][COL];
int i, j;

for (i = 0; i < ROW; i++) {

for (j = 0; j < COL; j++) { cellDetails[i][j].f = FLT_MAX; cellDetails[i][j].g = FLT_MAX; cellDetails[i][j].h = FLT_MAX;
cellDetails[i][j].parent_i = -1;
cellDetails[i][j].parent_j = -1;

}

}

```

```

i = src.first, j = src.second; cellDetails[i][j].f = 0.0;
cellDetails[i][j].g = 0.0;

cellDetails[i][j].h = 0.0; cellDetails[i][j].parent_i = i; cellDetails[i][j].parent_j = j; <f, <i, j>> where f = g + h,
and i, j are the row and column index of that cell Note that 0 <= i <= ROW-1 & 0 <= j <= COL-1 This open list is
implemented as a set of pair of pair.*/
set<pPair> openList;

// Put the starting cell on the open list and set its

// 'f' as 0

openList.insert(make_pair(0.0, make_pair(i, j)));

// the destination is not reached. bool foundDest = false;

while (!openList.empty()) {

pPair p = *openList.begin();

// Remove this vertex from the open list openList.erase(openList.begin());
i = p.second.first;

j = p.second.second; closedList[i][j] = true; Cell-->Popped Cell (i, j)

```

```

N --> North      (i-1, j)

S --> South      (i+1, j)

E --> East       (i, j+1)

W --> West       (i, j-1)

N.E--> North-East (i-1, j+1)

N.W--> North-West (i-1, j-1) S.E--> South-East (i+1, j+1) S.W--> South-West (i+1, j-1)*/
// To store the 'g', 'h' and 'f' of the 8 successors double gNew, hNew, fNew;
//----- 1st Successor (North) -----

// Only process this cell if this is a valid one if (isValid(i - 1, j) == true) {
// If the destination cell is the same as the

// current successor

if (isDestination(i - 1, j, dest) == true) {

// Set the Parent of the destination cell cellDetails[i - 1][j].parent_i = i; cellDetails[i - 1][j].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);
foundDest = true; return;
}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

```

```

// Else do the following

else if (closedList[i - 1][j] == false

&& isUnBlocked(grid, i - 1, j)

== true) { gNew = cellDetails[i][j].g + 1.0; hNew = calculateHValue(i - 1, j, dest); fNew = gNew + hNew;

// If it isn't on the open list, add it to

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell

//      OR

// If it is on the open list already, check

// to see if this path to that square is

// better, using 'f' cost as the measure. if (cellDetails[i - 1][j].f == FLT_MAX
|| cellDetails[i - 1][j].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i - 1, j)));

// Update the details of this cell cellDetails[i - 1][j].f = fNew; cellDetails[i - 1][j].g = gNew; cellDetails[i - 1][j].h = hNew;
cellDetails[i - 1][j].parent_i = i;

```

```

cellDetails[i - 1][j].parent_j = j;

}

}

}

//----- 2nd Successor (South) -----

// Only process this cell if this is a valid one if (isValid(i + 1, j) == true) {
// If the destination cell is the same as the

// current successor

if (isDestination(i + 1, j, dest) == true) {

// Set the Parent of the destination cell cellDetails[i + 1][j].parent_i = i; cellDetails[i + 1][j].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);
foundDest = true; return;
}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

// Else do the following

else if (closedList[i + 1][j] == false

&& isUnBlocked(grid, i + 1, j)
== true) { gNew = cellDetails[i][j].g + 1.0;

```

```

hNew = calculateHValue(i + 1, j, dest); fNew = gNew + hNew;
// If it isn't on the open list, add it to

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell

//      OR

// If it is on the open list already, check

// to see if this path to that square is

// better, using 'f' cost as the measure. if (cellDetails[i + 1][j].f == FLT_MAX
|| cellDetails[i + 1][j].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i + 1, j)));

// Update the details of this cell cellDetails[i + 1][j].f = fNew; cellDetails[i + 1][j].g = gNew; cellDetails[i + 1][j].h = hNew;
cellDetails[i + 1][j].parent_i = i; cellDetails[i + 1][j].parent_j = j;
}

}

}

//----- 3rd Successor (East) -----
// Only process this cell if this is a valid one if (isValid(i, j + 1) == true) {

```

```

// If the destination cell is the same as the
// current successor

if (isDestination(i, j + 1, dest) == true) {

// Set the Parent of the destination cell cellDetails[i][j + 1].parent_i = i; cellDetails[i][j + 1].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);
foundDest = true; return;
}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

// Else do the following

else if (closedList[i][j + 1] == false

&& isUnBlocked(grid, i, j + 1)

== true) { gNew = cellDetails[i][j].g + 1.0;
hNew = calculateHValue(i, j + 1, dest); fNew = gNew + hNew;
// If it isn't on the open list, add it to

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell

//      OR

// If it is on the open list already, check

```



```

// to see if this path to that square is

// better, using 'f' cost as the measure. if (cellDetails[i][j + 1].f == FLT_MAX
// cellDetails[i][j + 1].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i, j + 1)));

// Update the details of this cell cellDetails[i][j + 1].f = fNew; cellDetails[i][j + 1].g = gNew; cellDetails[i][j + 1].h = hNew;
cellDetails[i][j + 1].parent_i = i; cellDetails[i][j + 1].parent_j = j;
}

}

}

//----- 4th Successor (West) -----

// Only process this cell if this is a valid one if (isValid(i, j - 1) == true) {
// If the destination cell is the same as the

// current successor

if (isDestination(i, j - 1, dest) == true) {
// Set the Parent of the destination cell cellDetails[i][j - 1].parent_i = i; cellDetails[i][j - 1].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);

```

```

foundDest = true; return;
}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

// Else do the following

else if (closedList[i][j - 1] == false

&& isUnBlocked(grid, i, j - 1)

== true) { gNew = cellDetails[i][j].g + 1.0; hNew = calculateHValue(i, j - 1, dest); fNew = gNew + hNew;
// If it isn't on the open list, add it to

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell OR
// If it is on the open list already, check

// to see if this path to that square is

// better, using 'f' cost as the measure. if (cellDetails[i][j - 1].f == FLT_MAX
|| cellDetails[i][j - 1].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i, j - 1)));
// Update the details of this cell cellDetails[i][j - 1].f = fNew;

```

```

cellDetails[i][j - 1].g = gNew; cellDetails[i][j - 1].h = hNew; cellDetails[i][j - 1].parent_i = i; cellDetails[i][j - 1].parent_j = j;
}

}

}

//      5th Successor (North-East)

//.....

// Only process this cell if this is a valid one if (isValid(i - 1, j + 1) == true) {
// If the destination cell is the same as the

// current successor

if (isDestination(i - 1, j + 1, dest) == true) {

// Set the Parent of the destination cell cellDetails[i - 1][j + 1].parent_i = i; cellDetails[i - 1][j + 1].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);
foundDest = true; return;
}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

// Else do the following

else if (closedList[i - 1][j + 1] == false

```

```

&& isUnBlocked(grid, i - 1, j + 1)

== true) { gNew = cellDetails[i][j].g + 1.414;
hNew = calculateHValue(i - 1, j + 1, dest); fNew = gNew + hNew
// If it isn't on the open list, add it to

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell

//      OR

// If it is on the open list already, check

// to see if this path to that square is

// better, using 'f' cost as the measure.

if (cellDetails[i - 1][j + 1].f == FLT_MAX

|| cellDetails[i - 1][j + 1].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i - 1, j + 1)));

// Update the details of this cell cellDetails[i - 1][j + 1].f = fNew; cellDetails[i - 1][j + 1].g = gNew; cellDetails[i - 1][j + 1].h
= hNew; cellDetails[i - 1][j + 1].parent_i = i; cellDetails[i - 1][j + 1].parent_j = j;
}

}

```

```

}

//      6th Successor (North-West)

//.....

// Only process this cell if this is a valid one if (isValid(i - 1, j - 1) == true) {
// If the destination cell is the same as the

// current successor

if (isDestination(i - 1, j - 1, dest) == true) {

// Set the Parent of the destination cell cellDetails[i - 1][j - 1].parent_i = i; cellDetails[i - 1][j - 1].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);
foundDest = true; return;
}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

// Else do the following

else if (closedList[i - 1][j - 1] == false

&& isUnBlocked(grid, i - 1, j - 1)

== true) { gNew = cellDetails[i][j].g + 1.414;
hNew = calculateHValue(i - 1, j - 1, dest); fNew = gNew + hNew;
// If it isn't on the open list, add it to

```

```

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell

//      OR

// If it is on the open list already, check

// to see if this path to that square is

// better, using 'f' cost as the measure.

if (cellDetails[i - 1][j - 1].f == FLT_MAX

|| cellDetails[i - 1][j - 1].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i - 1, j - 1)));

// Update the details of this cell cellDetails[i - 1][j - 1].f = fNew; cellDetails[i - 1][j - 1].g = gNew; cellDetails[i - 1][j - 1].h =
hNew; cellDetails[i - 1][j - 1].parent_i = i; cellDetails[i - 1][j - 1].parent_j = j;
}

}

}

//      7th Successor (South-East)

//.....

// Only process this cell if this is a valid one if (isValid(i + 1, j + 1) == true) {
// If the destination cell is the same as the

// current successor

```

```

if (isDestination(i + 1, j + 1, dest) == true) {

// Set the Parent of the destination cell cellDetails[i + 1][j + 1].parent_i = i; cellDetails[i + 1][j + 1].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);
foundDest = true; return;
}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

// Else do the following

else if (closedList[i + 1][j + 1] == false

&& isUnBlocked(grid, i + 1, j + 1)

== true) { gNew = cellDetails[i][j].g + 1.414;
hNew = calculateHValue(i + 1, j + 1, dest); fNew = gNew + hNew
// If it isn't on the open list, add it to

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell

//      OR

// If it is on the open list already, check

// to see if this path to that square is

// better, using 'f' cost as the measure.

```

```

if (cellDetails[i + 1][j + 1].f == FLT_MAX

|| cellDetails[i + 1][j + 1].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i + 1, j + 1)));

// Update the details of this cell cellDetails[i + 1][j + 1].f = fNew; cellDetails[i + 1][j + 1].g = gNew; cellDetails[i + 1][j +
1].h = hNew; cellDetails[i + 1][j + 1].parent_i = i; cellDetails[i + 1][j + 1].parent_j = j;
}

}

}

//      8th Successor (South-West)

//.....

// Only process this cell if this is a valid one if (isValid(i + 1, j - 1) == true) {
// If the destination cell is the same as the

// current successor

if (isDestination(i + 1, j - 1, dest) == true) {

// Set the Parent of the destination cell cellDetails[i + 1][j - 1].parent_i = i; cellDetails[i + 1][j - 1].parent_j = j; printf("The
destination cell is found\n"); tracePath(cellDetails, dest);
foundDest = true;

```



```

return;

}

// If the successor is already on the closed

// list or if it is blocked, then ignore it.

// Else do the following

else if (closedList[i + 1][j - 1] == false

&& isUnBlocked(grid, i + 1, j - 1)

== true) { gNew = cellDetails[i][j].g + 1.414;
hNew = calculateHValue(i + 1, j - 1, dest); fNew = gNew + hNew;
// If it isn't on the open list, add it to

// the open list. Make the current square

// the parent of this square. Record the

// f, g, and h costs of the square cell

//      OR

// If it is on the open list already, check

// to see if this path to that square is

// better, using 'f' cost as the measure.

if (cellDetails[i + 1][j - 1].f == FLT_MAX

|| cellDetails[i + 1][j - 1].f > fNew) { openList.insert(make_pair(
fNew, make_pair(i + 1, j - 1)));

// Update the details of this cell cellDetails[i + 1][j - 1].f = fNew; cellDetails[i + 1][j - 1].g = gNew;

```

```

cellDetails[i + 1][j - 1].h = hNew; cellDetails[i + 1][j - 1].parent_i = i; cellDetails[i + 1][j - 1].parent_j = j;
}

}

}

}

// When the destination cell is not found and the open

// list is empty, then we conclude that we failed to

// reach the destination cell. This may happen when the

// there is no way to destination cell (due to

// blockages)

if (foundDest == false)

printf("Failed to find the Destination Cell\n");

return;

} // Driver program to test above function int main()
{

/* Description of the Grid- 1--> The cell is not blocked 0--> The cell is blocked */ int grid[ROW][COL]
= { { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },

{ 1, 1, 1, 0, 1, 1, 1, 0, 1, 1 },

{ 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },

{ 0, 0, 1, 0, 1, 0, 0, 0, 0, 1 },

```

```
{ 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },  
{ 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },  
{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 },  
{ 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },  
{ 1, 1, 1, 0, 0, 0, 1, 0, 0, 1 } };
```

```
// Source is the left-most bottom-most corner Pair src = make_pair(8, 0)  
// Destination is the left-most top-most corner Pair dest = make_pair(0, 0)  
aStarSearch(grid, src, dest); return (0);  
}
```

OUTPUT :

```
input  
The destination cell is found  
  
The Path is -> (8,0) -> (7,0) -> (6,0) -> (5,0) -> (4,1) -> (3,2) -> (2,1) -> (1,0) -> (0,0)  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

CONCLUSION: Thus, I have executed program to implement A* Path finding Algorithm

PRACTICAL 5

Aim : - Implement $N(N = 8)$ Queen's problem using Back Tracking and Explain What is Backpropagation.

Theory : -

Backtracking is a general algorithm for finding all (or some) solutions to some computational problem, that incrementally builds candidates to the solutions, and abandons each partial candidate 'c' ("backtracks") as soon as it determines that 'c' cannot possibly be completed to a valid solution. Backtracking is an important tool for solving constraint satisfaction problems, such as crosswords, verbal arithmetic, Sudoku, and many other puzzles. The n-queens problem consists of placing n queens on an $n \times n$ checker board in such a way that they do not threaten each other, according to the rules of the game of chess. Every queen on a checker square can reach the other squares that are located on the same horizontal, vertical, and diagonal line. So there can be at most one queen at each horizontal line, at most one queen at each vertical line, and at most one queen at each of the $4n-2$ diagonal lines. Furthermore, since we want to place as many queens as possible, namely exactly n queens, there must be exactly one queen at each horizontal line and at each vertical line. The concept behind backtracking algorithm which is used to solve this problem is to successively place the queens in columns. When it is impossible to place a queen in a column (it is on the same diagonal, row, or column as another token), the algorithm backtracks and adjusts a preceding queen.

For example :arrange 4 queen First option

	Q		
			Q
Q			
		Q	

Second option

		Q	
Q			
			Q
	Q		

1. Each recursive call attempts to place a queen in a specific column. For a given call, the state of the board from previous placements is known (i.e. where are the other queens?)

2. Current step backtracking: If a placement within the column does not lead to a solution, the queen is removed and moved "down" the column

3. Previous step backtracking: When all rows in a column have been tried, the call terminates and backtracks to the previous call (in the previous column)

4. If a queen cannot be placed into column i , do not even try to place one onto column $i+1$ rather, backtrack to column $i-1$ and move the queen that had been placed there. Using this approach we can reduce the number of potential solutions even more

Algorithm

1) Start in the leftmost column

2) If all queens are placed return true

3) Try all rows in the current column. Do following for every tried row.

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

3) If all rows have been tried and nothing worked, return false to trigger Backtracking

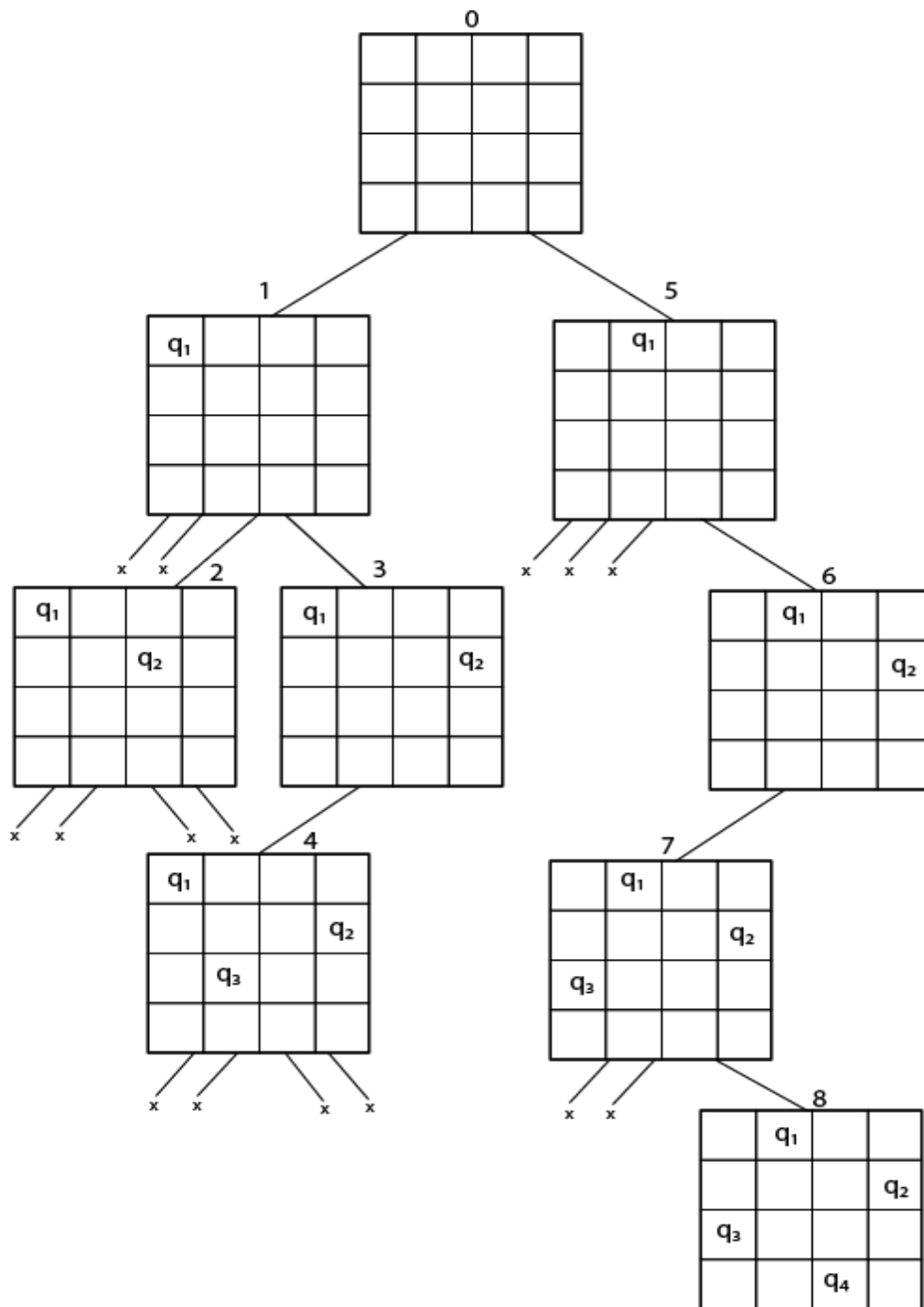


Fig shows the complete state space for 4 - queens problem. But we can use backtracking method to generate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.

Algorithm NQueens (k, n)

//Using backtracking, this procedure prints all possible placements of n queens

//on an n x n chessboard so that they are non-attacking

```
{
    for i ← 1 to n do
    {
        if(Place(k,i) )
        {
            x[k] ← i
            if (k=n)
                write ( x[1...n])
            else
                Nqueens (k+1, n)
        }
    }
}
```

Algorithm Place(k, i)

//Returns true if a queen can be placed in kth row and ith column. Otherwise it

//returns false. x[] is a global array whose first (k-1) values have been set. Abs(r)

//returns the absolute value of r.

```
{
    for j ← 1 to k-1 do
    {
        if ( (x[j]=i or Abs(x[j]-i) = Abs(j-k) )
        {
            return false
        }
    }
}
```

SOURCE CODE:-

```
#inclu
de
<iostr
eam>
#inclu
de
<vect
or>
using
names
pace
std;
bool isSafe(vector<vector<int>>& board, int row, int col, int N) {
```

```
// Check if there is a queen in  
the same columnfor (int i = 0; i  
< row; i++) {  
    if (board[i][col] == 1) {
```



```

        return false;

    }

}

// Check upper left diagonal

for (int i = row, j = col; i >= 0 && j
    >= 0; i--, j--) { if (board[i][j] ==
    1) {
        return false;

    }

}

// Check upper right diagonal

for (int i = row, j = col; i >= 0 && j
    < N; i--, j++) { if (board[i][j] ==
    1) {
        return false;

    }

}

return true;

}

bool solveNQueens(vector<vector<int>>& board,
    int row, int N) { if (row == N) {
    // All queens have been
    placed successfullyreturn
    true;
}

for (int col = 0; col
    < N; col++) { if
    (isSafe(board,

```

```
row, col, N)) {  
    board[row][col] = 1; // Place the queen
```

```

        if (solveNQueens(board,
            row + 1, N)) {return
            true;
        }

        // Backtrack if placing the queen
        leads to a solutionboard[row][col] =
        0;
    }

    return false;
}

void printBoard(const
    vector<vector<int>>& board) {int N
    = board.size();
    for (int i = 0;
        i < N; i++)
        { for (int j
            = 0; j < N;
            j++) {
            cout << board[i][j] << " ";

            }

            cout << endl;

        }
    }

}

int main() {

    int N = 8; // Number of queens and board size

```

```
// Create an empty chessboard  
vector<vector<int>> board(N,  
vector<int>(N, 0));
```

```

if
    (solveNQueens(boa
rd, 0, N)) { cout <<
    "Solution found:"
    << endl;
    printBoard(board);
} else {

    cout << "No solution exists." << endl;

}

return 0;

}

```

OUTPUT:

```

19 }
20
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
[Running] cd "c:\Users\HP\OneDrive\Desktop\AI PRACTICALS\" && g++ Backtracking.cpp -o Backtracking && "c:\
Solution found:
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0

[Done] exited with code=0 in 2.598 seconds

```

BACKPROPAGATION:

Backpropagation is an algorithm used in artificial intelligence (AI) to fine-tune mathematical weight functions and improve the accuracy of an artificial neural network's outputs.

A neural network can be thought of as a group of connected input/output (I/O) nodes. The level of accuracy each node produces is expressed as a loss function (error rate).

Backpropagation calculates the mathematical gradient of a loss function with respect to the other weights in the neural network. The calculations are then used to give artificial network nodes with high error rates less weight than nodes with lower error rates.

Backpropagation uses a methodology called chain rule to improve outputs. Basically, after each forward pass through a network, the algorithm performs a backward pass to adjust the model's weights.

An important goal of backpropagation is to give data scientists insight into how changing a weight function will change loss functions and the overall behavior of the neural network. The term is sometimes used as a synonym for "error correction."

CONCLUSION:

Thus, I have successfully implemented 8 Queens Problem in Python and Explained Back Propagation.

PRACTICAL 6

AIM: Implement Use Case of Supervised Learning

Use Case: OBJECT DETECTION

Theory:

Supervised Learning is a machine learning paradigm for acquiring the input-output relationship information of a system based on a given set of paired input-output training samples. As the output is regarded as the label of the input data or the supervision, an input-output training sample is also called labeled training data, or supervised data. Occasionally, it is also referred to as Learning with a Teacher Learning from Labeled Data, or Inductive Machine Learning. The goal of supervised learning is to build an artificial system that can learn the mapping between the input and the output, and can predict the output of the system given new inputs.

Object detection using TensorFlow involves leveraging pre-trained models such as Faster R-CNN, SSD, or YOLO. These models have been trained on extensive datasets like COCO or ImageNet and are capable of detecting objects across various classes. By utilizing these pre-trained models, the implementation process becomes more streamlined.

First, you need to select an appropriate pre-trained model based on factors like accuracy, speed, and available computing resources. TensorFlow provides a model zoo that offers a variety of pre-trained models to choose from. Once you have selected a model, you can load it by either downloading the model weights and configuration files or using TensorFlow's model zoo directly. The model files are typically available in formats like TensorFlow Saved Model, frozen graph (.pb) files, or checkpoints.

After loading the model, you need to preprocess the input images to match the model's requirements. This involves resizing the images to the model's input size, normalizing pixel values, and converting them to the appropriate color space. TensorFlow provides various functions and APIs to handle these preprocessing tasks efficiently.

Next, you can perform inference using the pre-trained model on the preprocessed input images. This involves passing the images through the model to obtain predictions, which include bounding box coordinates, class labels, and confidence scores for detected

objects. TensorFlow provides convenient APIs and functions to facilitate this inference process.

Once the inference is completed, you can apply post-processing techniques to refine the object detections. Common post-processing steps include filtering out low-confidence predictions, performing non-maximum suppression (NMS) to eliminate overlapping bounding boxes, and applying thresholding to remove detections below a certain confidence threshold.

Finally, to visualize the object detection results, you can draw bounding boxes around the detected objects on the input images and label them with class names and confidence scores. This visualization step helps in understanding and analyzing the performance of the object detection model.

It's important to note that specific implementation details may vary depending on the chosen pre-trained model and the object detection library being used.

TensorFlow offers comprehensive documentation and tutorials that provide code samples, pre-trained models, and guidance for fine-tuning models to suit specific object detection tasks.

Source Code :

```
import tensorflow as tf
import cv2
import matplotlib.pyplot as plt

# Load the pre-trained model
model = tf.keras.applications.MobileNetV2(weights='imagenet')

# Load the image
image_path = 'C:/Users/HP/Downloads/clock.jpg'
image = cv2.imread(image_path)

# Preprocess the image
# Preprocess the image
resized_image = cv2.resize(image, (224, 224))
input_image = tf.keras.applications.mobilenet.preprocess_input(resized_image)

# Add an extra dimension to the input image for batch size
input_image = tf.expand_dims(input_image, axis=0)

# Run object detection
predictions = model.predict(input_image)
predicted_labels = tf.keras.applications.mobilenet.decode_predictions(predictions, top=5)[0]

# Display the results
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title('Object Detection')
for label in predicted_labels:
    _, class_name, confidence = label
    plt.text(10, 30, f'{class_name}: {confidence:.2%}', color='red', fontsize=12, weight='bold')
plt.show()
```


Input



Output

Object Detection
analog_clock: 47.18%



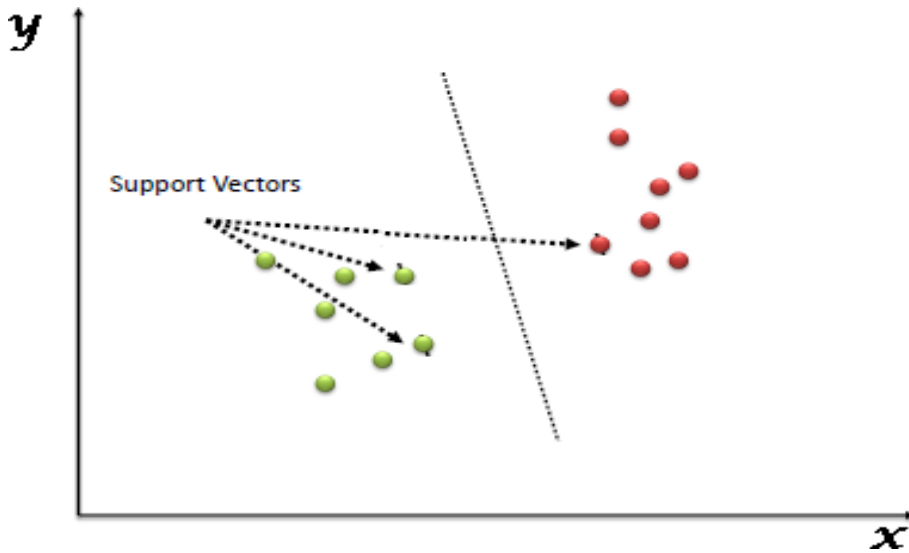
CONCLUSION: Thus, I have successfully executed an example of Supervised Learning- Object Detection using TensorFlow.

PRACTICAL 7

AIM: Implementation of Support Vector Machine (Weather Forecasting).

Theory:

“Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n- dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



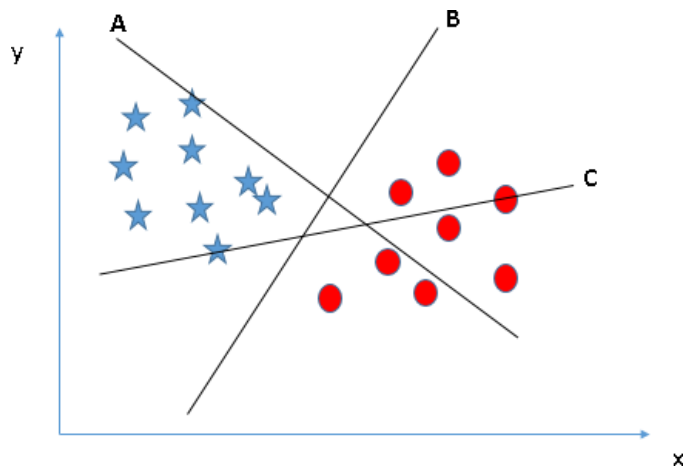
Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

How does it work?

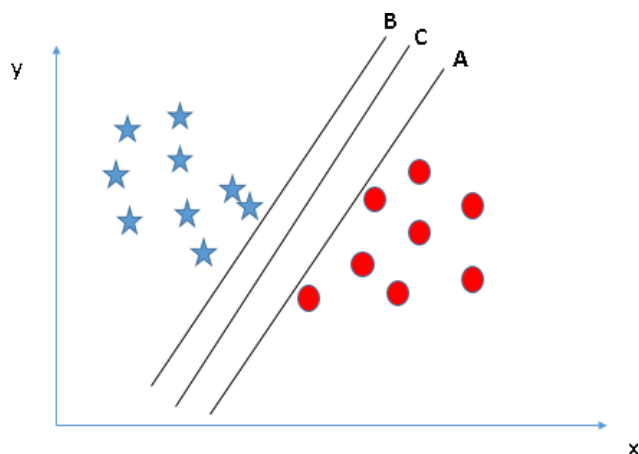
Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is “How can we identify the right hyper-plane?”. Don’t worry, it’s not as hard as you think!

Let’s understand:

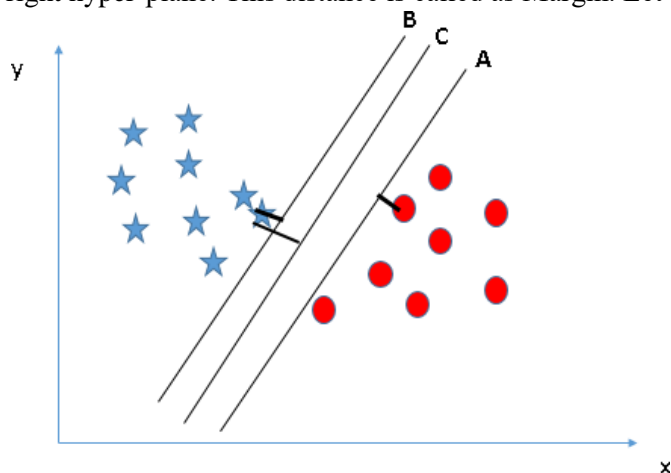
- Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B, and C). Now, identify the right hyper-plane to classify stars and circles.



- You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.
- Identify the right hyper-plane (Scenario-2): Here, we have three hyper-planes (A, B, and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

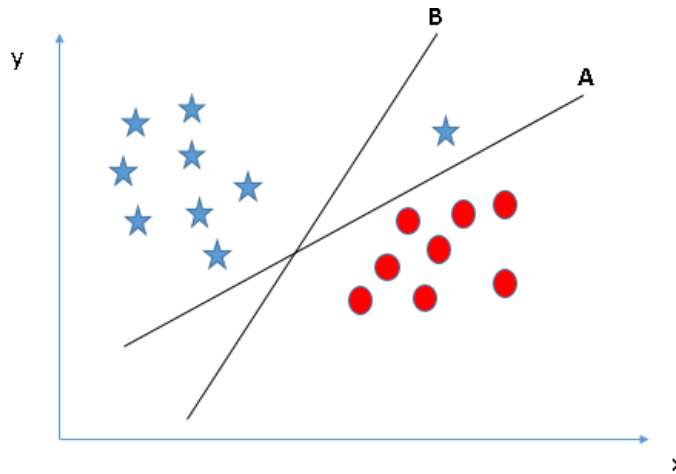


Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as Margin. Let's look at



Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper- plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

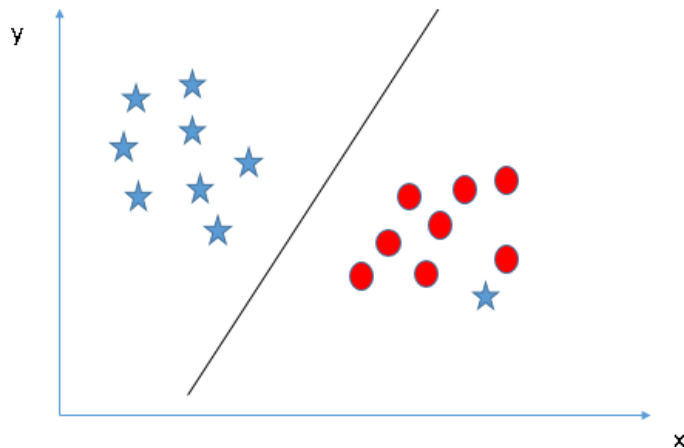
•Identify the right hyper-plane (Scenario-3):Hint: Use the rules as discussed in previous section to identify the right hyper-plane



Some of you may have selected the hyper-plane B as it has higher margin compared to A. But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is A.



•As I have already mentioned, one star at other end is like an outlier for star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.



Source Code:

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

# Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)
```

Conclusion:

I can conclude that I have successfully completed the execution of Support Vector Machine (Weather Forecasting).

PRACTICAL 8

AIM: Artificial Intelligence Lab Mini Project.

Theory :

Clustering is an unsupervised machine learning technique that aims to group similar data points together. In this case, the clustering is performed on two features: income and purchase rating.

The K-means algorithm is an iterative algorithm that aims to partition the data into K distinct clusters, where K is a predefined number. The algorithm works by randomly initializing K cluster centroids and then iteratively assigning data points to their nearest centroid and updating the centroids based on the assigned data points. This process continues until convergence, where the centroids no longer change significantly.

The elbow method is a common technique used to determine the optimal number of clusters for K-means. It involves plotting the within-cluster sum of squares (WCSS) against the number of clusters. WCSS represents the sum of squared distances between each data point and its assigned centroid within a cluster. The plot helps identify the "elbow point," which is the number of clusters where the decrease in WCSS slows down significantly. This point is often considered as a good choice for the number of clusters.

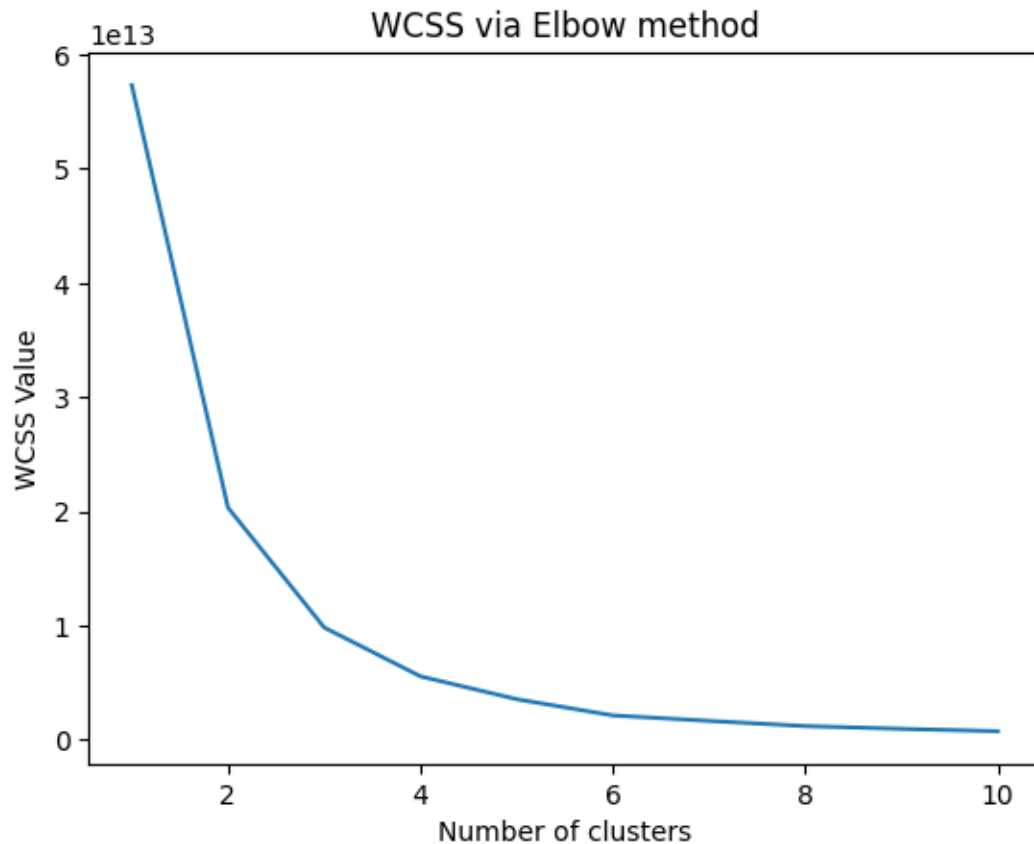
1. K-means Algorithm:

- K-means is an iterative algorithm that aims to partition data points into K clusters, where K is a user-defined parameter.
- The algorithm proceeds as follows:
 1. Randomly initialize K cluster centroids.
 2. Assign each data point to the nearest centroid based on a distance metric (usually Euclidean distance).
 3. Update the centroids by calculating the mean of all data points assigned to each cluster.
 4. Repeat steps 2 and 3 until convergence (when the centroids no longer change significantly or a maximum number of iterations is reached).
- K-means seeks to minimize the within-cluster sum of squares (WCSS), which represents the sum of squared distances between each data point and its assigned centroid within a cluster.
- The algorithm can be sensitive to the initial random centroid initialization, which can lead to different solutions. To mitigate this, the "k-means++" initialization method is often used, which intelligently selects initial centroids to improve convergence speed and reduce the chance of converging to suboptimal solutions.

2. Elbow Method:

- The elbow method is a technique used to determine the optimal number of clusters (K) for K-means clustering.
- It involves plotting the WCSS values against the number of clusters.
- The WCSS decreases as the number of clusters increases because adding more clusters allows data points to be assigned to centroids closer to them.
- However, as K continues to increase, the improvement in WCSS becomes less significant, resulting in a diminishing return.
- The elbow point in the plot represents the number of clusters where the rate of WCSS

improvement drastically decreases, forming an elbow-like shape. This point is often considered as a good choice for K.



3. Cluster Visualization:

- After performing K-means clustering, the resulting clusters can be visualized to gain insights and interpret the results.
- In a scatter plot, each data point is represented as a point, and each cluster is assigned a unique color or marker.
- Cluster centroids, which represent the average values of the data points within each cluster, are often marked as distinct symbols (e.g., crosses, circles) and plotted in the scatter plot.
- By visualizing the clusters, patterns, separations, and overlaps among the data points can be observed, providing a clearer understanding of the clustering results.

Results:

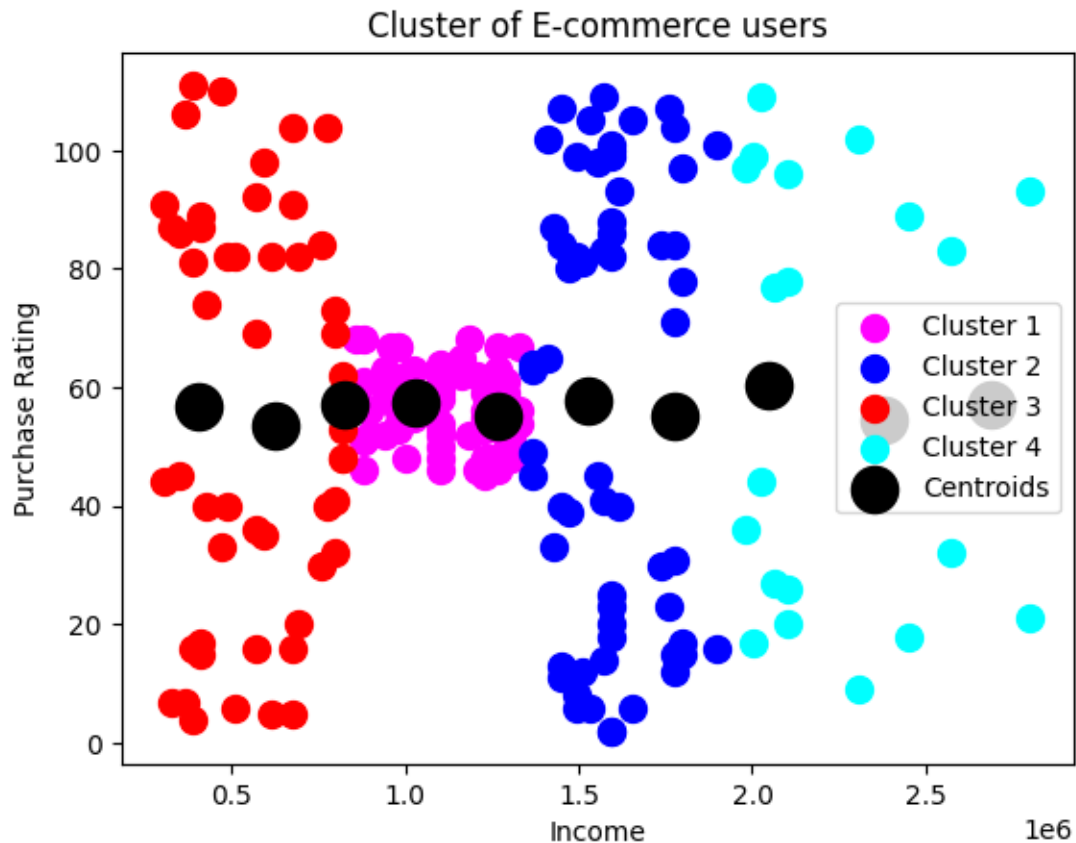
The code first reads the customer data from a CSV file and selects the 'Income' and 'Purchase Rating' columns as the features (X). It then performs K-means clustering with different numbers of clusters ranging from 1 to 10 and calculates the WCSS for each number of clusters. The WCSS values are stored in the list 'wcss'.

Next, the code plots the WCSS values against the number of clusters using matplotlib. The resulting plot helps visualize the elbow point.

After determining the optimal number of clusters, which is 4 in this case, the code performs K-means clustering

again with this number of clusters. It assigns each data point to its corresponding cluster and stores the cluster labels in 'y_means'.

Finally, the code visualizes the clusters by creating a scatter plot. Each cluster is represented by a different color, and the cluster centroids are marked with black crosses.



Conclusion :

The K-means clustering algorithm was applied to customer data consisting of income and purchase rating. By using the elbow method, the optimal number of clusters was determined to be 4. The resulting clusters were visualized in a scatter plot, showing distinct groups of customers based on their income and purchase rating.

This clustering analysis can provide valuable insights into customer segmentation, allowing businesses to target specific customer groups with tailored marketing strategies. The identified clusters can help in understanding customer behavior, identifying high-value customers, and making data-driven business decisions.