VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belagavi – 590018, Karnataka State, India

PROJECT ENTITLED

"INTERPOLATION, PREDICTION AND FEATURE ANALYSIS OF FINE-GRAINED AIR QUALITY"

Submitted in partial fulfilment of the requirements for the award of degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

For the academic year 2019-2020

Submitted by:

ABHIJNA S HEBBAR	**(1MV16CS002)**
AISHWARYA V	**(1MV16CS008)**
KULSUM BEGUM	**(1MV16CS045)**
NAMRATHA A	**(1MV16CS056)**

Project Carried out at

Sir M. Visvesvaraya Institute of Technology

Bengaluru - 562157

Under Guidance of

Dr. Rashmi Amardeep

Assistant Professor

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

HUNASAMARANAHALLI BENGALURU – 56215

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY

Krishnadevaraya Nagar, International Airport Road,
Hunasmaranahalli, Bengaluru – 562157

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

It is certified that the project work entitled "**INTERPOLATION, PREDICTION AND FEATURE ANALYSIS OF FINE-GRAINED AIR QUALITY**" is carried out by **ABHIJNA S HEBBAR (1MV16CS002), AISHWARYA V (1MV16CS008), KULSUM BEGUM (1MV16CS045), NAMRATHA A (1MV16CS056)** bonafide students of **Sir M Visvesvaraya Institute of Technology** in partial fulfilment for the award of the Degree of Bachelor of Engineering in Computer Science and Engineering of the **Visvesvaraya Technological University, Belagavi** during the year **2019-2020**. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the course of Bachelor of Engineering.

Name & Signature of Guide	Name & Signature of HOD	Name & Signature of Principal
Dr. Rashmi Amardeep Asst. Prof & Internal Guide Dept. Of CSE, Sir MVIT Bengaluru – 562157	**Dr. G. C. Bhanu Prakash** HOD, Dept. Of CSE, Sir MVIT Bengaluru - 562157	**Dr. V.R. Manjunath** Principal, Sir MVIT Bengaluru - 562157

External Examination:

Name of Examiner	Signature with Date
1)	
2)	

DECLARATION

We hereby declare that the entire project work embodied in this dissertation has been carried out by us and no part has been submitted for any degree or diploma of any institution previously.

Place: Bengaluru

Date:

Signature of Students:

ABHIJNA S HEBBAR	AISHWARYA V
(1MV16CS002)	(1MV16CS008)
KULSUM BEGUM	NAMRATHA A
(1MV16CS045)	(1MV16CS056)

ACKNOWLEDGEMENT

It gives us immense pleasure to express our sincere gratitude to the management of **Sir M. Visvesvaraya Institute of Technology,** Bengaluru for providing the opportunity and the resources to accomplish our project work in their premises.

On the path of learning, the presence of an experienced guide is indispensable and we would like to thank our guide **Dr. Rashmi Amardeep,** Assistant Professor, Dept. of CSE, for her invaluable help and guidance.

Heartfelt and sincere thanks to **Dr. G. C. Bhanu Prakash,** HOD, Dept. of CSE, for his suggestions, constant support and encouragement.

We would also like to convey our regards to **Dr. V.R. Manjunath,** Principal, Sir. MVIT for providing us with the infrastructure and facilities needed to develop our project.

We would also like to thank the staff of Department of Computer Science and Engineering and lab-in-charges for their co-operation and suggestions. Finally, we would like to thank all our friends for their help and suggestions without which completing this project would not have been possible.

- ABHIJNA S HEBBAR 1MV16CS002
- AISHWARYA V 1MV16CS008
- KULSUM BEGUM 1MV16CS045
- NAMRATHA A 1MV16CS056

ABSTRACT

The interpolation, prediction, and feature analysis of fine-gained air quality are three important topics in the area of urban air computing. The solutions to these topics can provide extremely useful information to support air pollution control, and consequently generate great societal and technical impacts. Most of the existing work solves the three problems separately by different models. This project proposes a general and effective approach to solve the three problems in one model. The proposed approach utilizes the information pertaining to the unlabeled spatiotemporal data to improve the performance of the interpolation and the prediction, and performs feature selection and association analysis to reveal the main relevant features of the air quality.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The interpolation, prediction, and feature analysis of fine-gained air quality are three important topics in the area of urban air computing. A good interpolation solves the problem that there are limited air-quality-monitor-stations whose distribution is uneven in a city; a precise prediction. Provides valuable information to protect humans from being damaged by air pollution; a reasonable feature analysis reveals the main relevant factors to the variation of air quality. In general, the solutions to these topics can extract extremely useful information to support air pollution control, and consequently generate great societal and technical impacts. The interpolation, prediction, and feature analysis of fine-gained air quality are three important topics in the area of urban air computing. The solutions to these topics can provide extremely useful information to support air pollution control, and consequently generate great societal and technical impacts. Most of the existing work solves the three problems separately by different models. This paper proposes a general and effective approach to solve the three problems in one model called the Deep Air Learning. The main idea of DAL lies in embedding feature selection and semisupervised learning in different layers of the deep learning network. The proposed approach utilizes the information pertaining to the unlabeled spatio -temporal data to improve the performance of the interpolation and the prediction, and performs feature selection and association analysis to reveal the main relevant features to the variation of the air quality.

CHAPTER 2

LITERARY SURVEY

1. **Spatiotemporal Interpolation Methods for Air Pollution Exposure**- investigates spatiotemporal interpolation methods for the application of air pollution assessment. The air pollutant of interest in this paper is fine particulate matter PM2.5. The choice of the time scale is investigated when applying the shape function-based method. It is found that the measurement scale of the time dimension has an impact on the interpolation results. Based upon the comparison between the accuracies of interpolation results, the most effective time scale out of four experimental ones was selected for per- forming the PM2.5interpolation. The paper also evaluates the population exposure to the ambient air pollution of PM2.5 at the county-level in the contiguous U.S. in 2009. The interpolated county- level PM2.5has been linked to 2009 population data and the population with a risky PM2.5exposure has been estimated. The risky PM2.5 exposure means the PM2.5 concentration exceeding the National Ambient Air Quality Standards. The geographic distribution of the counties with a risky PM2.5 exposure is visualized. This work is essential to understanding the associations between ambient air pollution exposure and population health outcomes.

2. **U-Air: When Urban Air Quality Inference Meets Big Data** - gives information about two separated classifiers. One is a spatial classifier based on an artificial neural network (ANN), which takes spatially- related features (e.g., the density of POIs and length of highways) as input to model the spatial correlation between air qualities of different locations. The other is a temporal classifier based on a linear-chain conditional random field (CRF), involving temporally-related features (e.g., traffic and meteorology) to model the temporal dependency of air quality in a location.

3. **Inferring Air Quality for Station Location Recommendation Based on Urban Big Data**- tries to answer two questions. First, how to infer real-time air quality of any arbitrary location given environmental data and data from very sparse monitoring locations. Second, if one needs to establish few new monitor stations to improve the inference quality, how to determine the best locations for such purpose? The problems are challenging since for most of the locations (>99%) they do not have any airquality data to train a model from. They de-sign a semi-supervised inference model utilizing existing

monitoring data together with heterogeneous city dynamics, including meteorology, human mobility, structure of road networks, and point of interests (POIs). They also propose an entropy- minimization model to suggest the best locations to establish new monitoring stations. They evaluate the proposed approach using Beijing air quality data, resulting in clear advantages over a series of state-of- the-art and commonly used methods.

4. **Model for Forecasting Expressway Fine Particulate Matter and Carbon Monoxide Concentration: Application of Regression and Neural Network Models**- The Bormann Expressway is a heavily traveled 16-mi segment of the Interstate 80/94 freeway through Northwestern Indiana. The Lake and Porter counties through which this expressway passes are designated as particulate matter 2.5m (PM2.5) and ozone 8-hr standard nonattainment areas. The Purdue University air quality group has been collecting PM2.5 , carbon monoxide (CO), wind speed, wind direction, pressure, and temperature data since September 1999. In this paper[4], regression and neural network models were developed for forecasting hourly PM2.5and CO concentrations. Time series of PM2.5and CO concentrations, traffic data, and meteorological parameters were used for developing the neural network and regression models. The models were compared using a number of statistical quality indicators. Both models had reasonable accuracy in predicting hourly PM2.5concentration with coefficient of determination 0.80, root mean square error (RMSE)4g/m 3, and index of agreement (IA) 0.90. For CO prediction, both models showed moderate fore-casting performance with a coefficient of determination 0.55, RMSE0.50 ppm, and IA0.85. These models are computationally less cumbersome and require less number of predictors as compared with the deterministic models.

5. **Forecasting Fine- Grained Air Quality Based on Big Data** - they forecast the reading of an air quality monitoring station over the next 48 hours, using a data-driven method that considers current meteorological data, weather forecasts, and air quality data of the station and that of other stations within a few hundred kilometers. Our predictive model is comprised of four major components: 1) a linear regression-based temporal predictor to model the local factors of air quality, 2) a neural network-based spatial predictor to model global factors, 3) a dynamic aggregator combining the predictions of the spatial and temporal predictors according to meteorological data, and 4) an inflection predictor to capture sudden changes in air quality. They evaluate our model with data from 43 cities in China, surpassing the results of multiple baseline methods. They have deployed a system with the Chinese

Ministry of Environmental Protection, providing 48-hour fine-grained air quality forecasts for four major Chinese cities every hour. The forecast function is also enabled on Microsoft Bing Map and MS cloud platform Azure. Our technology is general and can be applied globally for other cities.

6. **Graph-Based Semi- Supervised Learning With Multi-Label** – Conventional graph-based semi-supervised learning methods predominantly focus on single label problem. However, it is more popular in real-world applications that an example is associated with multiple labels simultaneously. This paper proposes a novel graph- based learning framework in the setting of semi-supervised learning with multilabel. The proposed approach is characterized by simultaneously exploiting the inherent correlations among multiple labels and the label consistency over the graph. They apply the pro-posed framework to video annotation and report superior performance compared to key existing approaches over the TRECVID 2006 corpus.

7. **Combining Labeled and Unlabeled Data with Co-Training -** considers the problem of using a large unlabeled sample to boost performance of a learning algorithm when only a small set of labeled examples is available. In particular, they consider a problem setting motivated by the task of learning to classify web pages, in which the description of each example can be partitioned into two distinct views. For example, the description of a web page can be partitioned into the words occurring on that page, and the words occurring in hyperlinks that point to that page. They assume that either view of the example would be sufficient for learning if they had enough labeled data, but our goal is to use both views together to allow inexpensive unlabeled data to augment, a much smaller set of labeled examples. Specifically, the presence of two distinct views of each example suggests strategies in which two learning algorithms are trained separately on each view, and then each algorithm's predictions on new unlabeled examples are used to enlarge the training set of the other. Our goal in this paper is to provide a PAC-style analysis for this setting, and, more broadly, a PAC-style framework for the general problem of learning from both labeled and unlabeled data. They also provide empirical results on real web-page data indicating that this use of unlabeled examples can lead to significant improvement of hypotheses in practice.

8. **Co-training for Predicting Emotions with Spoken Dialogue Data** – Natural Language Processing applications often require large amounts of annotated training data, which are expensive to obtain. This

paper investigates the applicability of Co-training to train classifiers that predict emotions in spoken dialogues. In order to do so, they have first applied the wrapper approach with Forward Selection and Naïve Bayes, to reduce the dimensionality of our feature set. Our results show that Co- training can be highly effective when a good set of features are chosen.

9. **Learning with Limited and Noisy Tagging** – With the rapid development of social networks, tagging has become an important means responsible for such rapid development. A robust tagging method must have the capability to meet the two challenging requirements: limited labeled training samples and noisy labeled training samples. This paper investigates this challenging problem of learning with limited and noisy tagging and propose a discriminative model, called SpSVM-MC, that exploits both labeled and unlabeled data through a semi-parametric regularization and takes advantage of the multi-label constraints into the optimization. While SpSVM-MC is a general method for learning with limited and noisy tagging, in the evaluations they focus on the specific application of noisy image tagging with limited labeled training samples on a benchmark dataset. Theoretical analysis and extensive evaluations in comparison with state-of-the-art literature demonstrate that SpSVM-MC outstands with a superior performance.

10. **Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions** - Introduces a novel machine learning frame-work based on recursive auto encoders for sentence-level prediction of sentiment label distributions. Our method learns vector space representations for multi-word phrases. In sentiment prediction tasks these representations outperform other state-of-the-art approaches on commonly used datasets, such as movie reviews, without using any pre-defined sentiment lexica or polarity shifting rules. They also evaluate the model's ability to predict sentiment distributions on a new dataset based on confessions from the experience project. The dataset consists of personal user stories annotated with multiple labels which, when aggregated, form a multinomial distribution that captures emotional reactions. Our algorithm can more accurately predict distributions over such labels compared to several competitive baselines.

CHAPTER 3

SYSTEM REQUIREMENTS AND SPECIFICATON

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

3.1 Development Requirements

3.1.1 Hardware Requirements

PROCESSOR	:	Intel i5
RAM	:	4 GB
HARD DISK	:	500 GB

Any desktop/Laptop system with above configuration or higher level.

3.1.2 Software Requirements

OPERATING SYSTEM	:	Windows 8/10
BACK-END	:	Python3
FRAME WORK	:	Anaconda
IDE	:	Jupyter Notebook
OTHER BACKEND LIBRARIES	:	Numpy, pandas, matplotlib, seaborn, scikitplot

CHAPTER 4

SYSTEM ANALYSIS

4.1 Existing System

There exists insufficient air-quality- monitor stations in a city due to the high cost of building and maintaining such a station, it is expensive to obtain labeled training samples when dealing with fine- gained air quality.

The interpolation, prediction, and feature analysis of fine-gained air quality are three important topics in the area of urban air computing. The solutions to these topics can provide extremely useful information to support air pollution control, and consequently generate great societal and technical impacts. Most of the existing work solves the three problems separately by different models.

4.1.1 Disadvantages of ExistingSystem

The reason for the incomplete labels is related to the air quality monitor devices. In general, each station only has one monitor device which needs to be maintained at intervals, thus there will be no outputs for the station when the device is being maintained, recalibrated, or has otherproblems.

4.2 Proposed System

This project is motivated to address all these challenges by utilizing the information contained in the unlabeled data and the spatio-temporal data, and performing feature selection and association analysis for the urban air related data. Though labeled data are difficult or expensive to obtain, large amounts of unlabeled examples can often be gathered cheaply. In general, unlabeled data can help in providing information to better exploit the geometric structure of the data.

4.2.1 Advantages of Proposed System:

The proposed feature selection and analysis method reveals the importance of different input features to the predictions of the neural networks, thus has the ability to reveal some inner mechanism of the black-box deep models, which does not limit to air pollution prevention and control, but can also be applied to many other applications, such as medical diagnosis and terrorism detection.

This project consists of the following steps:

- Data Preprocessing and Labeling
- Data Analysis
- Splitting and Modeling Data
- Predictions
- Air Quality Analysis

CHAPTER 5

SYSTEM DESIGN

5.1 System Architecture

A system architecture diagram would be used to show the relationship between different components. Usually they are created for systems which include hardware and software and these are represented in the diagram to show the interaction between them.

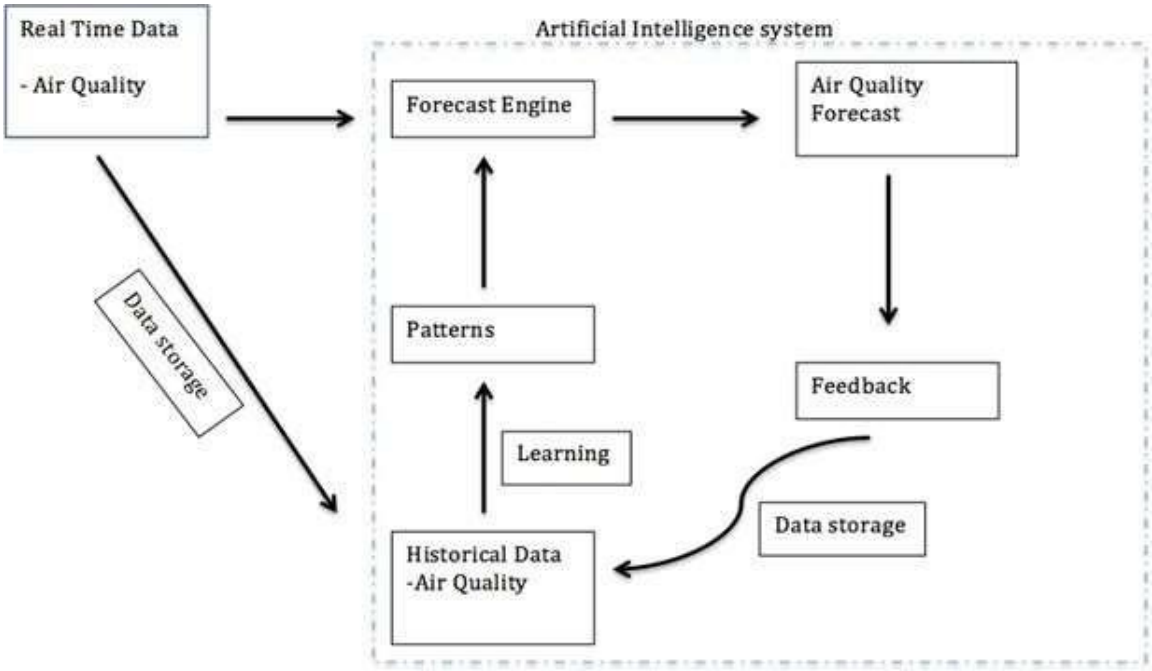


Figure 5.1 system architecture

5.2 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated.

5.2.1 DAFTAFLOW DIAGRAM LEVEL 0

Figure 5.2 Dataflow diagram level 0

5.2.2 DAFTAFLOW DIAGRAM LEVEL 1

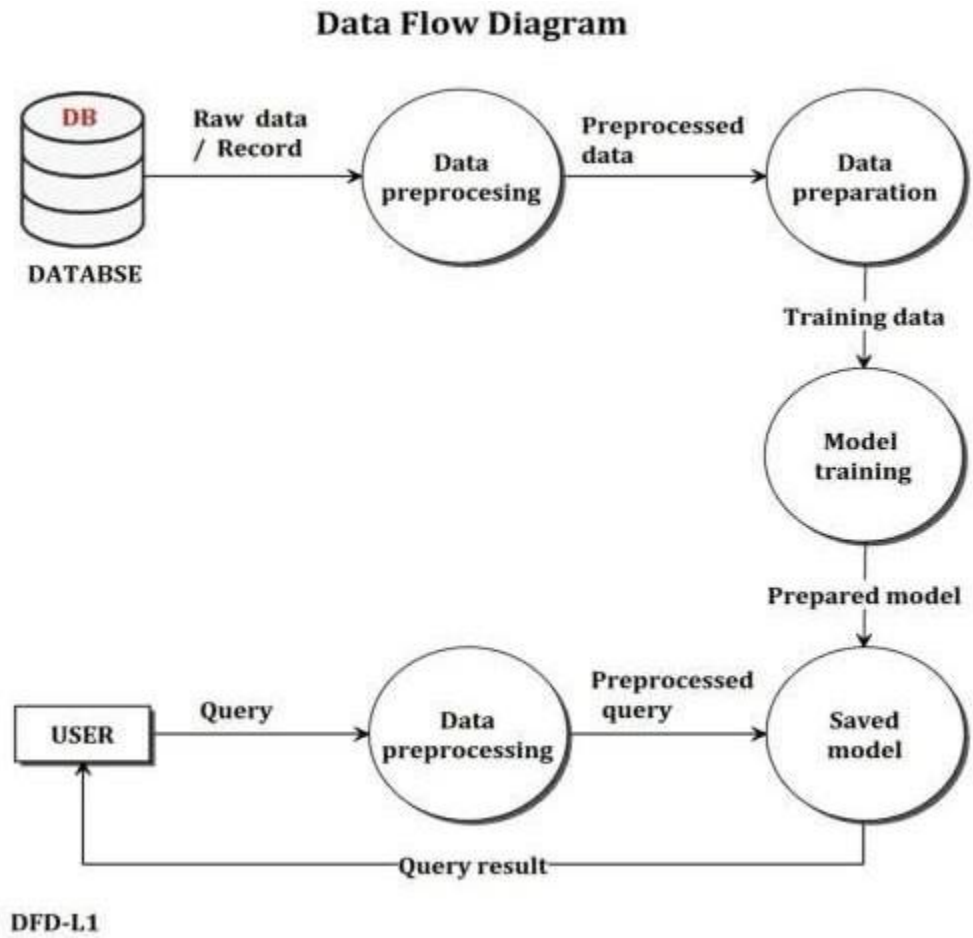


Figure 5.3 Dataflow diagram level 1

5.2.3 LEVEL2 DATA FLOW DIAGRAM:

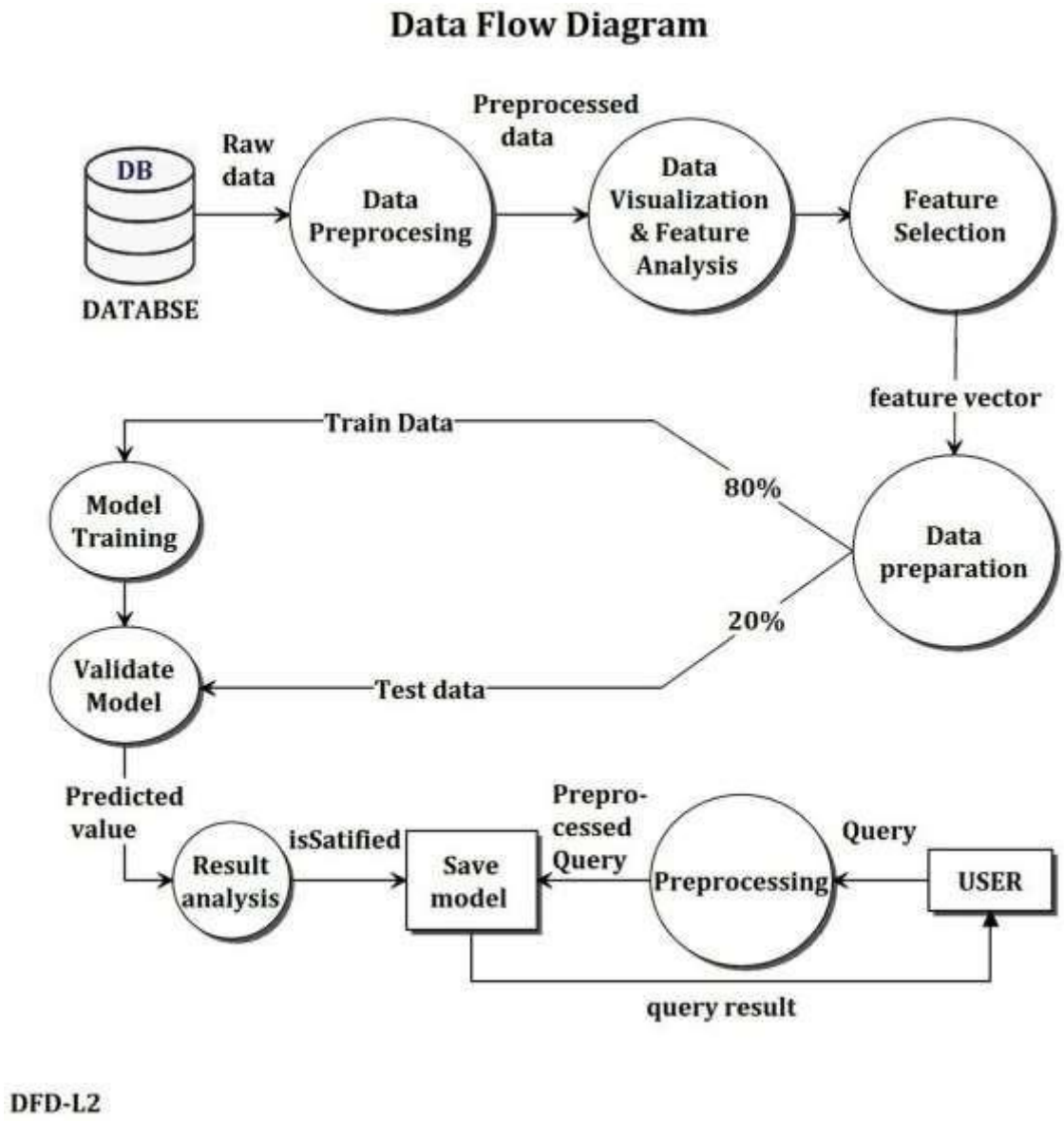


Fig 5.4 Level 2 Data Flow Diagram

5.3 FLOWCHART

A flowchart is one of the seven basic quality tools used in project management and it displays the actions that are necessary to meet the goals of a particular task in the most practical sequence. Also called as process maps, this type of tool displays a series of steps with branching possibilities that depict one or more inputs and transforms them to outputs.

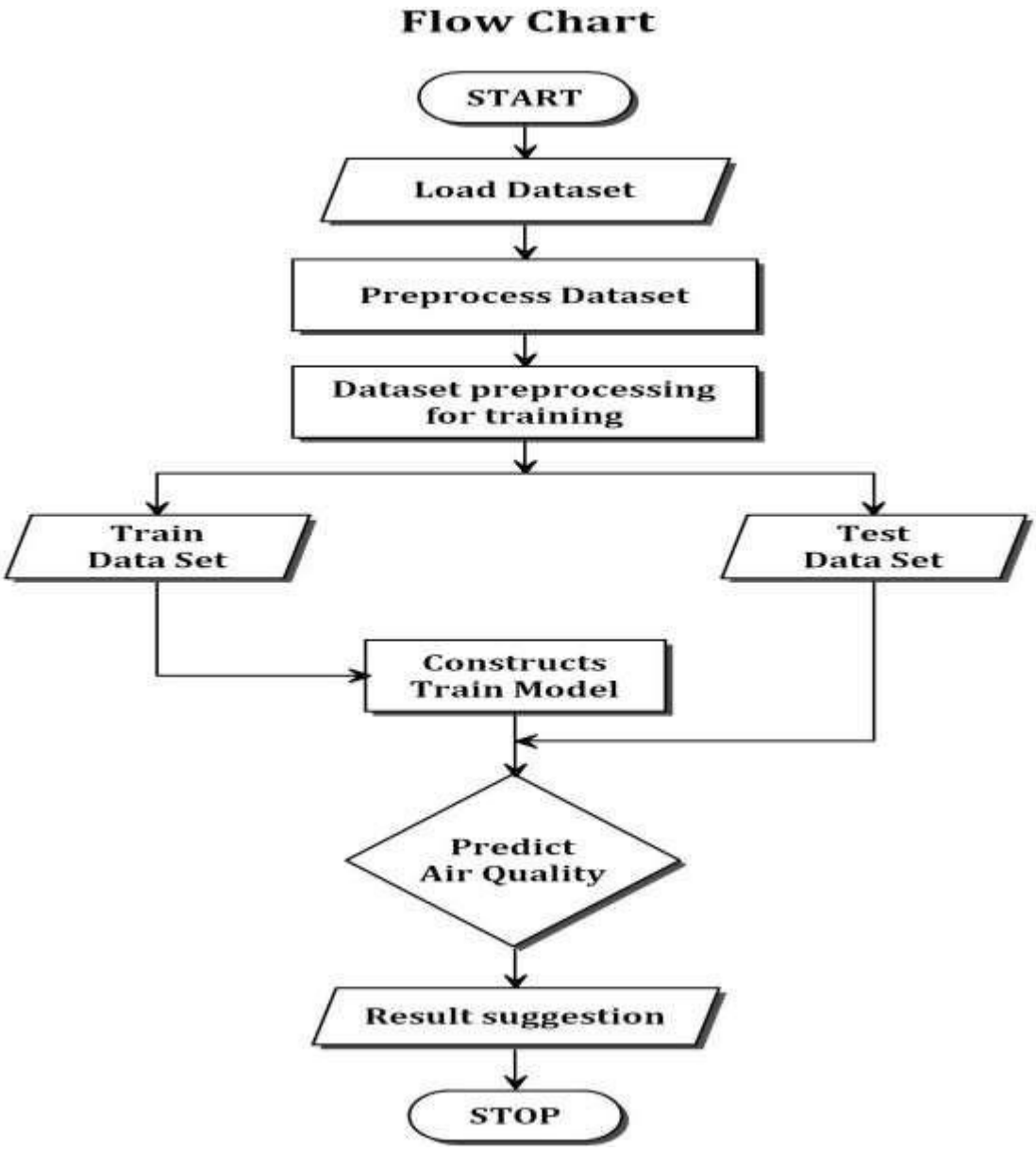


Figure 5.5 Flowchart

5.4 USECASE DIAGRAM

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

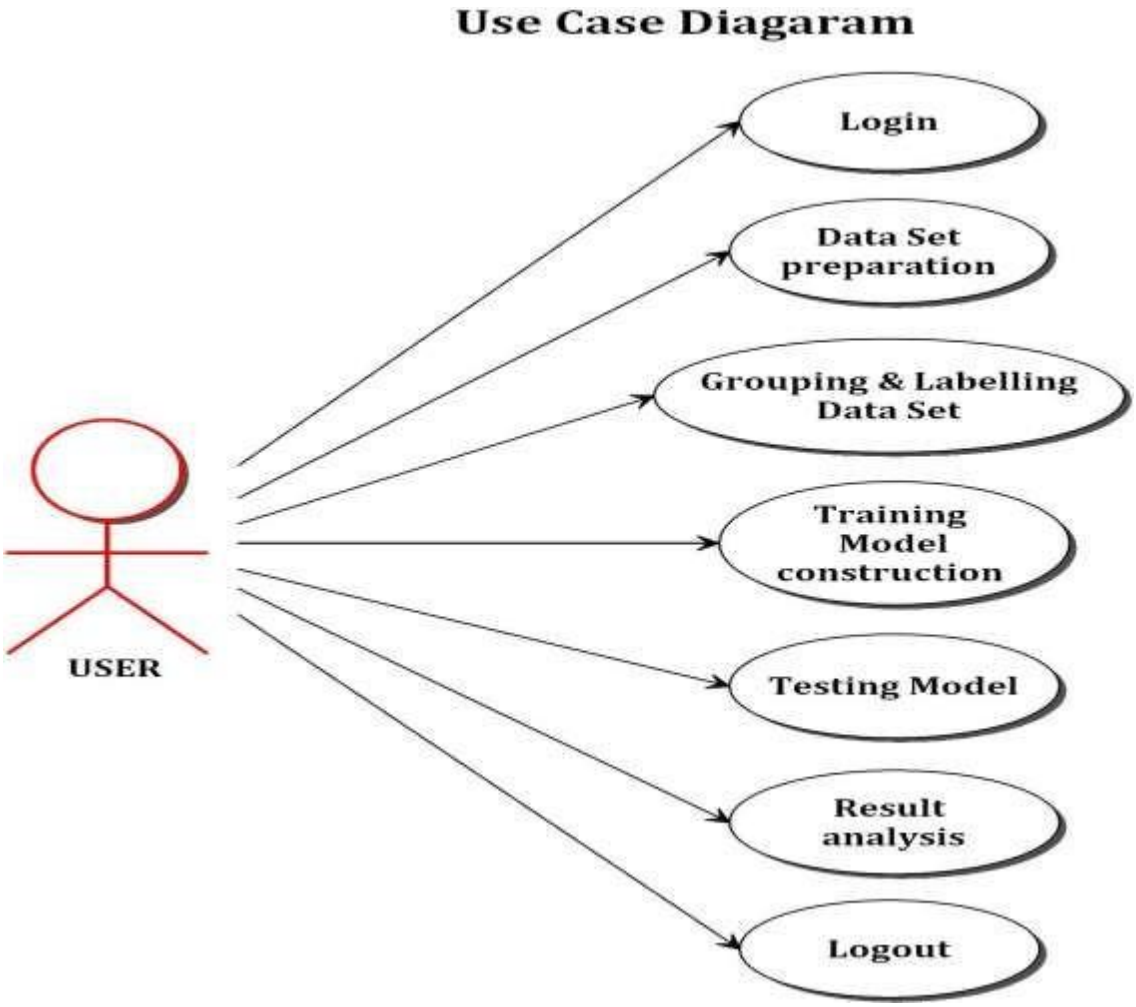


Figure 5.6 Use case diagram

CHAPTER 6

IMPLEMENTATION

6.1 PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

6.1.1 Features of Python

- **Easy-to-learn**: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read**: Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain**: Python's source code is fairly easy-to-maintain.
- **A broad standard library**: Python's bulk of the library is very portable and cross platform compatible on UNIX, Windows, and Macintosh.

- **Interactive Mode**: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases**: Python provides interfaces to all major commercial databases.
- **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable**: Python provides a better structure and support for large programs than shell scripting.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

6.2 MACHINE LEARNING

Machine Learning is a method of statistical learning where each instance in a dataset is described by a set of features or attributes. In contrast, the term “Deep Learning” is a method of statistical learning that extracts features or attributes from raw data. Deep Learning does this by utilizing neural networks with many hidden layers, big data, and powerful computational resources. The terms seem somewhat interchangeable, however, with Deep Learning methods, the algorithm constructs representations of the data automatically. In contrast, data representations are hard-coded as a set of features in machine learning algorithms, requiring further processes such as feature selection and extraction, (such as PCA).

Both of these terms are in dramatic contrast with another class of classical artificial intelligence algorithms known as Rule-Based Systems where each decision is manually programmed in such a way that it resembles a statistical model.

6.2.1 Semi-Supervised Machine Learning

Problems where you have a large amount of input data (X) and only some of the data is labeled (Y) are called semi-supervised learning problems. These problems sit in between both supervised and

unsupervised learning . A good example is a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.

Many real world machine learning problems fall into this area. This is because it can be expensive or time-consuming to label data as it may require access to domain experts. Whereas unlabeled data is cheap and easy to collect and store.

6.3 NumPy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. All this is explained with the help of examples for better understanding.

NumPy is a Python package which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array objects and a collection of routines for processing of arrays.

Numeric: The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

6.4 Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. Python has been great for data manipulation and preparation, but less so for data analysis and modeling , engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.

Pandas is the Python package providing fast, reliable, flexible, and expressive data structures designed to make working with 'relational' or 'labeled' data both easy and intuitive way.
Pandas aim to be the fundamental high-level building block for doing practical, real-world data modeling and

analysis in Python Programming Language.

6.4.1 Key Features of Pandas

The key features of Pandas are the following:

- Pandas library is a fast and efficient DataFrame object with the default and customized indexing.
- Pandas library helps for loading the data into in-memory data objects from different file formats.
- It has functions that deal with Data alignment and integrated the handling of missing data.
- Using Pandas, we can reshape and pivot the data sets.
- It has Label-based slicing, indexing, and subsetting of more massive datasets.
- Pandas can insert or delete the Columns from the data structure.
- We can use Pandas for data aggregation and transformations.
- It gives the High-performance merging and joining of data.
- Time Series functionality.

6.5 Matplotlib (Visualization)

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. . Matplotlib is a multi-platform data visualization library built on NumPy arrays, and designed to work with the broader SciPy stack. It was conceived by John Hunter in 2002.

6.5.1 Merits of Matplotlib

The idea behind Matplotlib can be summed up in the following motto as quoted by John Hunter, the creator and project leader of Matplotlib:

- It uses Python: Python is a very interesting language for scientific purposes (it's interpreted, high-level, easy to learn, easily extensible, and has a powerful standard library).
- It's open source, so no license to pay.This makes it very appealing for professors and students, who often have a low budget.
- It's a real programming language:
- It's much more complete: Python has a lot of external modules that will help us perform all the functions we need to..
- It's very customizable and extensible: Matplotlib can fit every use case because it has a lot of graph

types, features, and configuration options.

- It's integrated with LaTeX markup: This is really useful when writing scientific papers.
- It's cross-platform and portable: Matplotlib can run on Linux, Windows, Mac OS X, and Sun Solaris (and Python can run on almost every architecture available).

6.6 BACKENDS

A backend that displays the image on screen is called a user interface backend.The backend is that part of Matplotlib that works behind the scenes and allows the software to target several different output formats and GUI libraries(for screen visualization).

In order to be even more flexible, Matplotlib introduces the following wo layers structured (only for GUI output)

- The renderer :This actually does the drawing
- The canvas : This is the destination of the figure.

The standard renderer is the Anti-Grain Geometry (AGG) library, a high performance rendering engine which is able to create images of publication level quality , with anti-aliasing and sub pixel accuracy .AGG is responsible for the beautiful appearance of Matplotlib graphs. The canvas is provided with the GUI libraries , and any of them can use the AGG rendering ,along with the support for other rendering engines(for example ,GTK+).

6.7 S K LEARN

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas.

6.7.1 Components of scikit-learn:

Scikit-learn comes loaded with a lot of features.

- **Supervised learning algorithms:** Starting from Generalized linear models (e.g Linear Regression), Support Vector Machines (SVM), Decision Trees to Bayesian methods – all of them are part of scikit-

learn toolbox.

- **Cross-validation:** There are various methods to check the accuracy of supervised models on unseen data using sklearn.
- **Unsupervised learning algorithms:** Again there is a large spread of machine learning algorithms in the offering – starting from clustering, factor analysis, principal component analysis to unsupervised neural networks.
- **Various toy datasets:** This came in handy while learning scikit-learn. I had learned SAS using various academic datasets (e.g. IRIS dataset, Boston House prices dataset). Having them handy while learning a new library helped a lot.
- **Feature extraction:** Scikit-learn for extracting features from images and text (e.g. Bag of words)

CHAPTER 7

SOFTWARE IMPLEMENTATION

The project phase has two main elements namely data collection and the use of hardware and software in order to deploy semi-supervised learning algorithms to produce the desired results.

7.1 Data Acquisition and Preprocessing

Machine learning needs two things to work, data and models. When acquiring the data, we have to be sure to have enough data populated to train correctly the learning model. In general, the more data ,the better so make to come with enough rows.

The primary data collected from the online sources remains in the raw form of statements, digits and qualitative terms. The raw data contains error, omissions and inconsistencies. It requires corrections after careful scrutinizing the completed questionnaires. The following steps are involved in the processing of primary data. A huge volume of raw data collected through field survey needs to be grouped for similar details of individual responses.

Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for theanalysis.

Therefore, certain steps are executed to convert the data into a small clean data set. This technique is performed before the execution of Iterative Analysis.

Data Preprocessing is necessary because of the presence of unformatted real-world data. Mostly real-world data is composed of –

Inaccurate data (missing data) - There are many reasons for missing data such as data is not continuously collected, a mistake in data entry, technical problems with biometrics and much more. The rows that are found to contain missing data or missing values are hence dropped.

The presence of noisy data (erroneous data and outliers) - Thereasonsfor the existence of noisydata could be a technological problem of gadget that gathers data, a human mistake during data entry and much more. The presence of noisy data is eliminated by replacing them with other suitable values which are in accordance with the other values present in data set.

Inconsistent data - The presence of inconsistencies are due to the reasons such that existence of duplication within data, human data entry, containing mistakes in codes or names, i.e., violation of data constraints and much more.

7.2 Feature Selection and Data Preparation

Feature Selection is the process those features are automatically or manually selected which contribute most to the prediction variable or output in which interest lies in.

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process.

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. The feature selection technique used to construct this model is **Filter-based feature selection** method. Filter based feature selection methods use statistical measures to score the correlation or dependence between input variables that can be filtered to choose the most relevant features.

The process of organizing data into groups and classes on the basis of certain characteristics is known as the classification of data. Classification helps in making comparisons among the categories of observations. It can be either according to numerical characteristics or according to attributes. So here we need to visualize the prepared data to find whether the training data contains the correct label, which is known as a target or target attribute.

We slice a single data set into a training set and test set.

- Training set - a subset to train a model.
- Test set - a subset to test the trained model.

This is done by importing the module **train_test_split** from the library **sklearn.model_selection** The ratio of train:test is 80:20.

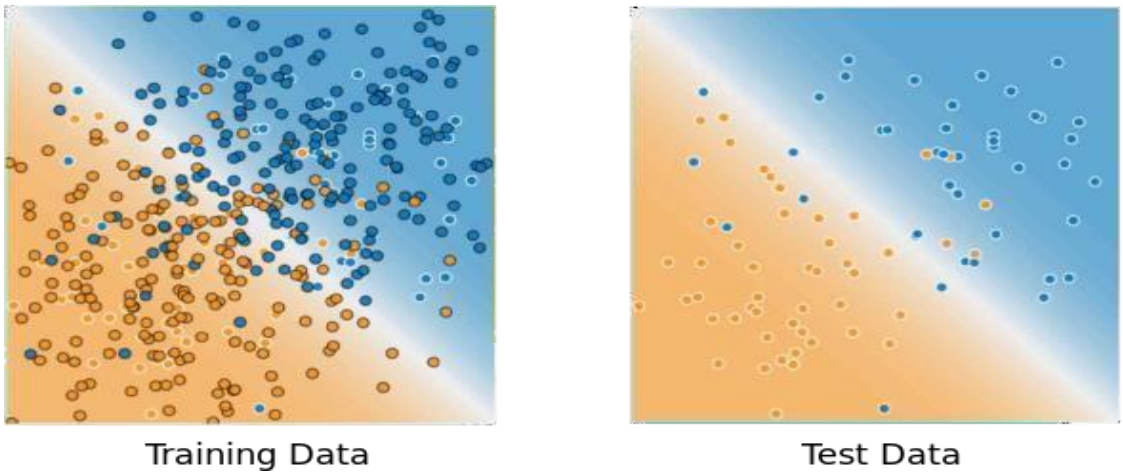


Fig : Classification of Training and test data

7.3 Model Construction and Model Training

The process of training an ML model involves providing an ML algorithm (that is, the learning algorithm) with training data to learn from. The term ML model refers to the model artifact that is created by the training process. The training data must contain the correct answer, which is known as a target or target attribute. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that is to be predicted), and it outputs an ML model that captures these patterns.

7.3.1 Naive Bayes Classifier

A Naive Bayes Classifier is a supervised machine-learning algorithm that uses the Bayes' Theorem, which assumes that features are statistically independent. The theorem relies on the naive assumption that input variables are independent of each other, i.e. there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results.

Naive Bayes Classifiers rely on the Bayes' Theorem, which is based on conditional probability or in simple terms, the likelihood that an event (A) will happen *given that* another event (B) has already happened. Essentially, the theorem allows a hypothesis to be updated each time new evidence is introduced. The equation below expresses Bayes' Theorem in the language of probability:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Where:

"P" is the symbol to denote probability.

P(A | B) = The probability of event A (hypothesis) occurring given that B (evidence) has occurred.
P(B | A) = The probability of the event B (evidence) occurring given that A (hypothesis) has occurred.
P(A) = The probability of event B (hypothesis) occurring.
P(B) = The probability of event A (evidence) occurring.

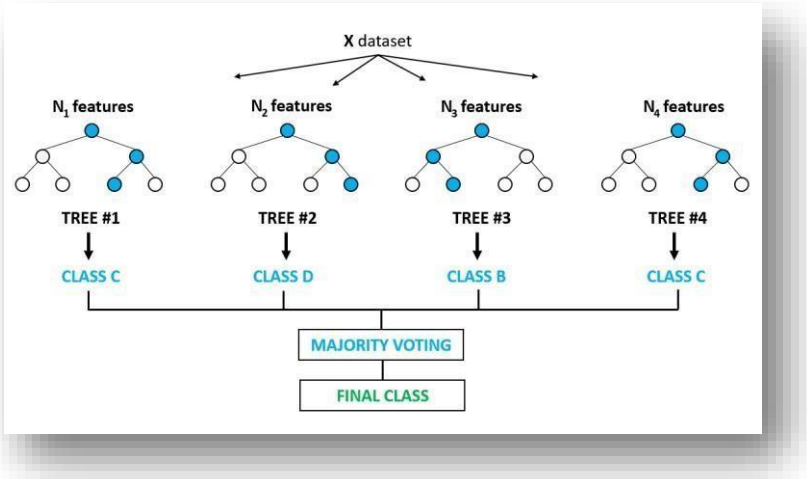
7.3.2 Random Forest Model

Random forest algorithm is a supervised classification algorithm. As the name suggest, this algorithm creates the forest with a number of trees. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

In the random forest classifier, the **higher the number** of trees in the forest gives **the high accuracy** results.

Random forest algorithm advantages:

- Random forest classifier will handle the missing values.
- When we have more trees in the forest, random forest classifier won't overfit the model.
- Can model the random forest classifier for categorical values also.



7.3.3 K Nearest Neighbors

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Distance functions

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

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

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

7.4 Model Validation

In testing phase the model is applied to new set of data. The training and test data are two different datasets. The goal in building a machine learning model is to have the model perform well. On the training set, as well as generalize well on new data in the test set**.**

CHAPTER 8

SOURCE CODE

```
In [1]: # import libraries
        import os
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import scikitplot as skplt

In [2]: # set input directory
        input_dir = 'input'

In [3]: # set dataset file name
        input_data_file = 'data.xlsx'

In [4]: # display list of files in input directory
        os.listdir('input')

Out[4]: ['airquality.csv', 'data.xlsx', 'labeled_data.csv']

In [5]: # to get the input data path
        data_path = os.path.join(os.curdir,input_dir,input_data_file)
        data_path

Out[5]: '.\\input\\data.xlsx'

In [6]: # to read data from excel file
        raw_data = pd.read_excel(data_path)
```

Fig 8.1 – import library

```
In [8]: raw_data.head()
```

Out[8]:

	TEMP	CH4	CO	NMHC	NO	NO2	NOx	O3	PM10	PM2.5	RH	SO2
0	16	2.1	0.79	0.14	1.2	16	17	37	177	78x	57	12
1	16	2.1	0.8	0.15	1.3	16	17	36	178	77x	57	11
2	16	2.1	0.71	0.13	1	13	14	38	163	72x	57	8
3	15	2	0.66	0.12	0.8	11	12	39	147	65x	58	6.5
4	15	2	0.53	0.11	0.6	10	11	38	131	56x	58	5.5

```
In [9]: raw_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 218639 entries, 0 to 218638
Data columns (total 12 columns):
TEMP     200169 non-null object
CH4      95822 non-null object
CO       217310 non-null object
NMHC     95614 non-null object
NO       217227 non-null object
NO2      216681 non-null object
NOx      217228 non-null object
O3       199864 non-null object
PM10     215761 non-null object
PM2.5    215768 non-null object
RH       200243 non-null object
SO2      217046 non-null object
dtypes: object(12)
memory usage: 20.0+ MB
```

Fig 8.2 – attributes

```
In [10]: raw_data.columns

Out[10]: Index(['TEMP', 'CH4', 'CO', 'NMHC', 'NO', 'NO2', 'NOx', 'O3', 'PM10', 'PM2.5',
                'RH', 'SO2'],
               dtype='object')

In [11]: raw_data.isna().sum()

Out[11]: TEMP      18470
         CH4      122817
         CO         1329
         NMHC     123025
         NO         1412
         NO2        1958
         NOx        1411
         O3        18775
         PM10       2878
         PM2.5      2871
         RH        18396
         SO2        1593
         dtype: int64
```

Fig 8.3 – missing values

```
In [13]: raw_data['O3'] = raw_data['O3'].fillna(raw_data['O3'].mean())
         raw_data['PM2.5'] = raw_data['PM2.5'].fillna(raw_data['PM2.5'].mean())
         raw_data['TEMP'] = raw_data['TEMP'].fillna(raw_data['TEMP'].mean())
         raw_data['CH4'] = raw_data['CH4'].fillna(raw_data['CH4'].mean())
         raw_data['CO'] = raw_data['CO'].fillna(raw_data['CO'].mean())
         raw_data['NMHC'] = raw_data['NMHC'].fillna(raw_data['NMHC'].mean())
         raw_data['NO'] = raw_data['NO'].fillna(raw_data['NO'].mean())
         raw_data['NO2'] = raw_data['NO2'].fillna(raw_data['NO2'].mean())
         raw_data['NOx'] = raw_data['NOx'].fillna(raw_data['NOx'].mean())
         raw_data['PM10'] = raw_data['PM10'].fillna(raw_data['PM10'].mean())
         raw_data['RH'] = raw_data['RH'].fillna(raw_data['RH'].mean())
         raw_data['SO2'] = raw_data['SO2'].fillna(raw_data['SO2'].mean())

In [14]: raw_data.isna().sum()

Out[14]: TEMP     0
         CH4      0
         CO       0
         NMHC     0
         NO       0
         NO2      0
         NOx      0
         O3       0
         PM10     0
         PM2.5    0
         RH       0
         SO2      0
         dtype: int64
```

Fig 8.4 – fill missing value rows

In [11]:
```
def numeric(row):
    try:
        row =str(row)
        row=(row.replace('x','').replace('#','').replace('*','').replace('NR',''))
        if (row==''):
            return
        else:
            return float(row.replace('x','').replace('#','').replace('*','').replace('NR',''))

    except TypeError:
        row =str(row)
        if (row==''):
            return
        else:
            return float(row.replace('x','').replace('#','').replace('*','').replace('NR',''))
```

In [12]:
```
raw_data['O3'] = raw_data['O3'].apply(numeric)
raw_data['PM2.5'] = raw_data['PM2.5'].apply(numeric)
raw_data['TEMP'] = raw_data['TEMP'].apply(numeric)
raw_data['CH4'] = raw_data['CH4'].apply(numeric)
raw_data['CO'] = raw_data['CO'].apply(numeric)
raw_data['NMHC'] = raw_data['NMHC'].apply(numeric)
raw_data['NO'] = raw_data['NO'].apply(numeric)
raw_data['NO2'] = raw_data['NO2'].apply(numeric)
raw_data['NOx'] = raw_data['NOx'].apply(numeric)
raw_data['PM10'] = raw_data['PM10'].apply(numeric)
raw_data['RH'] = raw_data['RH'].apply(numeric)
raw_data['SO2'] = raw_data['SO2'].apply(numeric)
```

Fig 8.5 – fix NaN

In [18]:
```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 0 to 5107
Data columns (total 12 columns):
TEMP     5000 non-null float64
CH4      5000 non-null float64
CO       5000 non-null float64
NMHC     5000 non-null float64
NO       5000 non-null float64
NO2      5000 non-null float64
NOx      5000 non-null float64
O3       5000 non-null float64
PM10     5000 non-null float64
PM2.5    5000 non-null float64
RH       5000 non-null float64
SO2      5000 non-null float64
dtypes: float64(12)
memory usage: 507.8 KB
```

Fig 8.6 – attributes

```
In [16]: raw_data.describe()
```

Out[16]:

	TEMP	CH4	CO	NMHC	NO	NO2	NOx	O3	PM10	PM2.5
count	218639.000000	218639.000000	218639.000000	218639.000000	218639.000000	218639.000000	218639.000000	218639.000000	218639.000000	218639.000000
mean	23.311751	1.928567	0.562539	0.267663	9.014298	17.882309	26.831967	29.047728	45.837390	19.521462
std	5.639131	0.158130	0.542817	0.158450	17.691158	12.030102	26.610134	17.112154	168.684043	59.907200
min	-30.000000	-0.800000	-0.890000	-8.240000	-9.400000	-9.200000	-4.700000	-8.500000	-396.000000	-57.000000
25%	19.000000	1.900000	0.270000	0.230000	1.400000	8.600000	10.000000	16.000000	26.000000	10.000000
50%	23.311751	1.928567	0.420000	0.267663	2.700000	16.000000	19.000000	29.047728	37.000000	16.000000
75%	27.000000	1.928567	0.660000	0.267663	7.600000	25.000000	33.000000	39.000000	52.000000	24.000000
max	59.000000	15.000000	38.000000	6.860000	358.000000	166.000000	411.000000	200.000000	9999.000000	10199.000000

Fig 8.7 – statistical data

```
In [20]: temp_data = raw_data[['TEMP','RH','CH4','NMHC','NO', 'NOx']]
```

```
In [21]: # axis = 1 : columnwise operation
         data = raw_data.drop(columns=['TEMP','RH','CH4','NMHC','NO', 'NOx'],axis=1)
         data.head()
```

Out[21]:

	CO	NO2	O3	PM10	PM2.5	SO2
0	0.79	16.0	37.0	177.0	78.0	12.0
1	0.80	16.0	36.0	178.0	77.0	11.0
2	0.71	13.0	38.0	163.0	72.0	8.0
3	0.66	11.0	39.0	147.0	65.0	6.5
4	0.53	10.0	38.0	131.0	56.0	5.5

Fig 8.8 – factors for labeling

```
In [22]: plt.figure(figsize=(20,10), dpi=80)
         image = plt.imread('img/aqi.png')
         plt.axis('off')
         plt.imshow(image)
         plt.show()
```

For the proposed AQI, a maximum operator system is selected:

$$AQI = Max(I_1, I_2, I_3, \ldots, I_n)$$

There are two reasons for adopting a maximum operator:

- Free from eclipsing and ambiguity (Ott 1978)
- Health effects of combination of pollutants (synergistic effects) are not known and thus a health-based index cannot be combined or weighted

Fig 8.9 – AQI calculation

```
In [23]: data['AQI']= data.max(axis=1)
         data.head()
```

Out[23]:

	CO	NO2	O3	PM10	PM2.5	SO2	AQI
0	0.79	16.0	37.0	177.0	78.0	12.0	177.0
1	0.80	16.0	36.0	178.0	77.0	11.0	178.0
2	0.71	13.0	38.0	163.0	72.0	8.0	163.0
3	0.66	11.0	39.0	147.0	65.0	6.5	147.0
4	0.53	10.0	38.0	131.0	56.0	5.5	131.0

```
In [24]: data['AQI'].describe()
```

```
Out[24]: count    5000.000000
         mean       53.106000
         std       101.548314
         min        10.000000
         25%        35.000000
         50%        45.000000
         75%        61.000000
         max      5004.000000
         Name: AQI, dtype: float64
```

Fig 8.10 – AQI index

```
In [25]: aqi = data['AQI']
         #aqi
```

```
In [26]: plt.figure(figsize=(12,5), dpi=60)
         image = plt.imread('img/table.png')
         plt.axis('off')
         plt.imshow(image)
         plt.show()
```

Table 3.2: IND-AQI Category and Range

AQI Category	AQI Range
Good	0 – 50
Satisfactory	51 – 100
Moderately-polluted	101 – 200
Poor	201 – 300
Very Poor	301 – 400
Severe	401 - 500

Fig 8.11 – AQI category

```
In [27]: plt.figure(figsize=(12,5), dpi=80)
         image = plt.imread('img/significance.png')
         plt.axis('off')
         plt.imshow(image)
         plt.show()
```

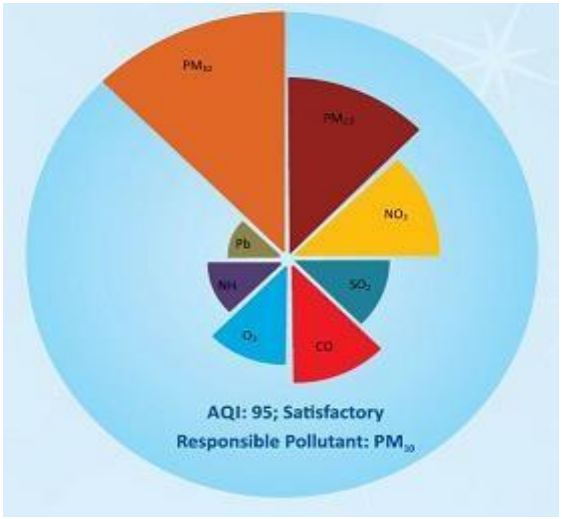


Fig 8.12 – AQI significance

```
In [28]: aqi[(aqi > 0) & (aqi <= 50)] = 0
         aqi[(aqi > 50) & (aqi <= 100)] = 1
         aqi[(aqi > 100) & (aqi <= 150)] = 2
         aqi[(aqi > 150) & (aqi <= 200)] = 3
         #aqi[(aqi > 200) & (aqi <= 300)] = 4
         aqi[(aqi > 200)] = 4

In [29]: aqi.unique()

Out[29]: array([3., 2., 1., 0., 4.])

In [30]: data['label'] = data['AQI'].astype('int8')

In [31]: data.drop(columns=['AQI'], axis= 1, inplace=True)

In [32]: data.tail()
```

Out[32]:

	CO	NO2	O3	PM10	PM2.5	SO2	label
5103	0.37	20.0	53.0	33.0	15.0	7.0	1
5104	0.51	34.0	55.0	42.0	21.0	21.0	1
5105	0.63	35.0	43.0	43.0	25.0	14.0	0
5106	0.59	27.0	43.0	40.0	26.0	3.6	0
5107	0.60	27.0	19.0	38.0	25.0	2.3	0

Fig 8.13 - Categorization

```
In [33]: temp_data.head()
```

Out[33]:

	TEMP	RH	CH4	NMHC	NO	NOx
0	16.0	57.0	2.1	0.14	1.2	17.0
1	16.0	57.0	2.1	0.15	1.3	17.0
2	16.0	57.0	2.1	0.13	1.0	14.0
3	15.0	58.0	2.0	0.12	0.8	12.0
4	15.0	58.0	2.0	0.11	0.6	11.0

```
In [34]: temp_data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 0 to 5107
Data columns (total 6 columns):
TEMP    5000 non-null float64
RH      5000 non-null float64
CH4     5000 non-null float64
NMHC    5000 non-null float64
NO      5000 non-null float64
NOx     5000 non-null float64
dtypes: float64(6)
memory usage: 273.4 KB
```

Fig 8.14 – other factors

```
In [35]: data.index

Out[35]: Int64Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
                     ...
                     5098, 5099, 5100, 5101, 5102, 5103, 5104, 5105, 5106, 5107],
                    dtype='int64', length=5000)

In [36]: data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 0 to 5107
Data columns (total 7 columns):
CO       5000 non-null float64
NO2      5000 non-null float64
O3       5000 non-null float64
PM10     5000 non-null float64
PM2.5    5000 non-null float64
SO2      5000 non-null float64
label    5000 non-null int8
dtypes: float64(6), int8(1)
memory usage: 278.3 KB

In [37]: data = temp_data.join(data)
```

Fig 8.15 – join data

In [38]: data.head()

Out[38]:

	TEMP	RH	CH4	NMHC	NO	NOx	CO	NO2	O3	PM10	PM2.5	SO2	label
0	16.0	57.0	2.1	0.14	1.2	17.0	0.79	16.0	37.0	177.0	78.0	12.0	3
1	16.0	57.0	2.1	0.15	1.3	17.0	0.80	16.0	36.0	178.0	77.0	11.0	3
2	16.0	57.0	2.1	0.13	1.0	14.0	0.71	13.0	38.0	163.0	72.0	8.0	3
3	15.0	58.0	2.0	0.12	0.8	12.0	0.66	11.0	39.0	147.0	65.0	6.5	2
4	15.0	58.0	2.0	0.11	0.6	11.0	0.53	10.0	38.0	131.0	56.0	5.5	2

```
In [39]: # save the labeled data
         data.to_csv('input/labeled_data.csv',index=False)
```

Data Preprocessing and Labeling Completed

Fig 8.16 – labelling

Data Analysis

```
In [40]: data.head()
```

Out[40]:

	TEMP	RH	CH4	NMHC	NO	NOx	CO	NO2	O3	PM10	PM2.5	SO2	label
0	16.0	57.0	2.1	0.14	1.2	17.0	0.79	16.0	37.0	177.0	78.0	12.0	3
1	16.0	57.0	2.1	0.15	1.3	17.0	0.80	16.0	36.0	178.0	77.0	11.0	3
2	16.0	57.0	2.1	0.13	1.0	14.0	0.71	13.0	38.0	163.0	72.0	8.0	3
3	15.0	58.0	2.0	0.12	0.8	12.0	0.66	11.0	39.0	147.0	65.0	6.5	2
4	15.0	58.0	2.0	0.11	0.6	11.0	0.53	10.0	38.0	131.0	56.0	5.5	2

Fig 8.17 – labelled data

```
In [41]: sns.pairplot(data)

Out[41]: <seaborn.axisgrid.PairGrid at 0x2c8cf2e0b00>
```

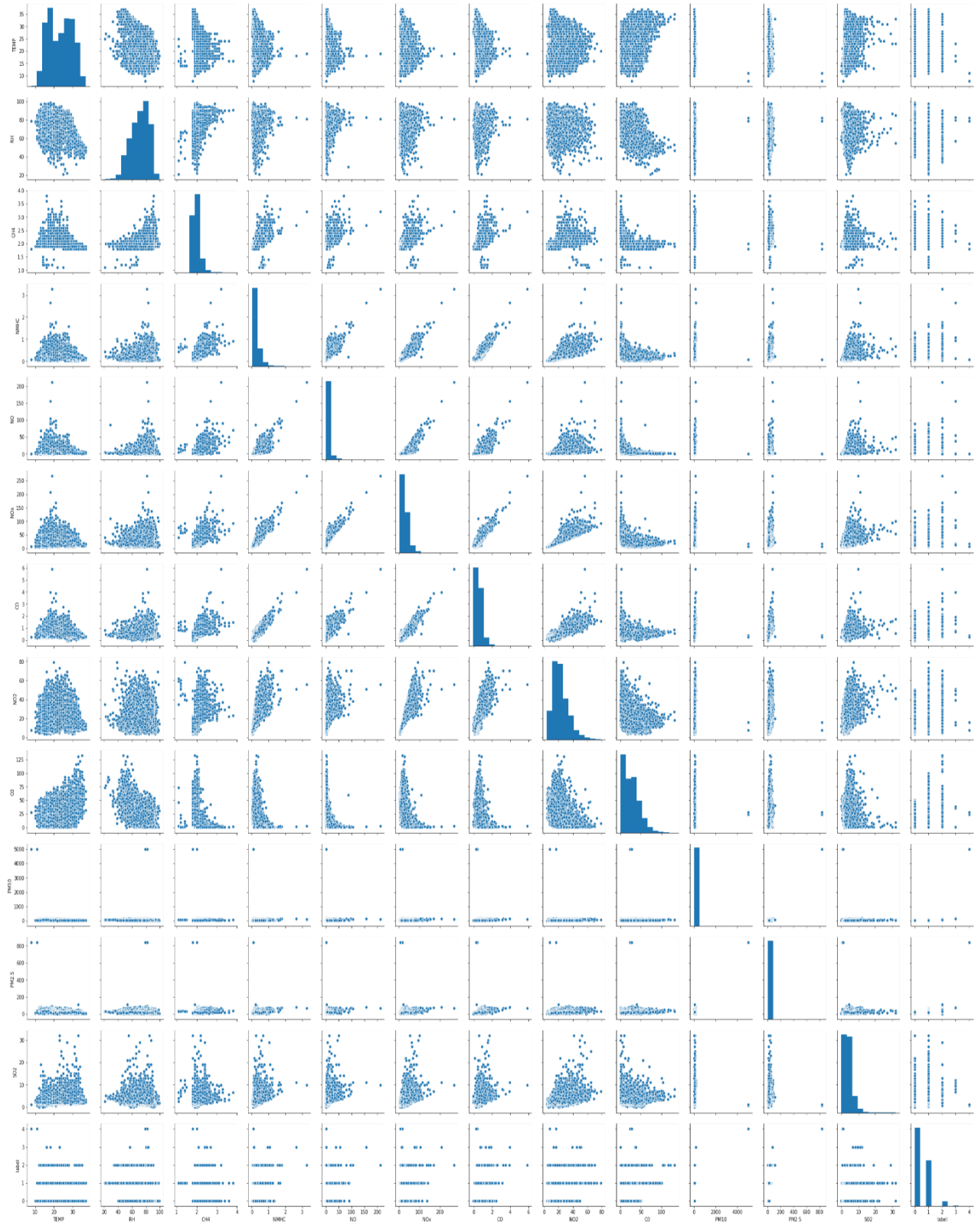


Fig 8.18 - pairplot

```
In [42]: sns.set(style="white", color_codes=True)
         sns.jointplot(x='PM2.5',y='RH',data=data)
Out[42]: <seaborn.axisgrid.JointGrid at 0x2c8cbc44908>
```

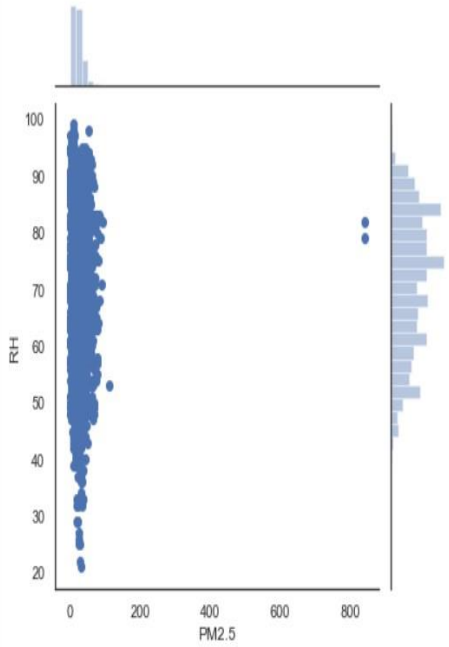


Fig 8.19 – jointplot1

```
In [43]: sns.set(style="white", color_codes=True)
         sns.jointplot(x='CH4',y='RH',data=data)
Out[43]: <seaborn.axisgrid.JointGrid at 0x2c8cc521b00>
```

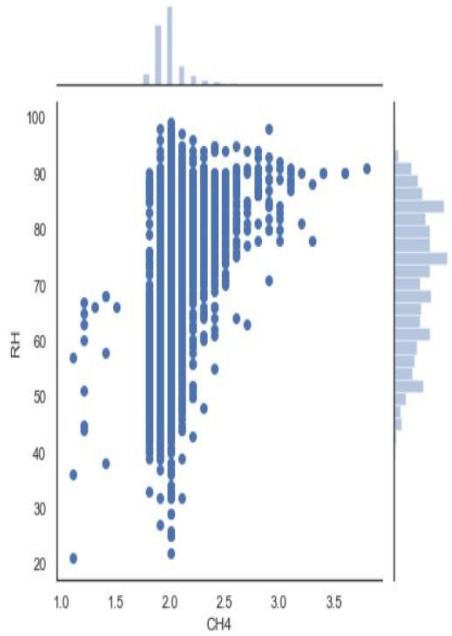


Fig 8.20 – jointplot2

```
In [44]: sns.set(style="white", color_codes=True)
         sns.jointplot(x='CH4',y='NOx',data=data)

Out[44]: <seaborn.axisgrid.JointGrid at 0x2c8cc801d30>
```

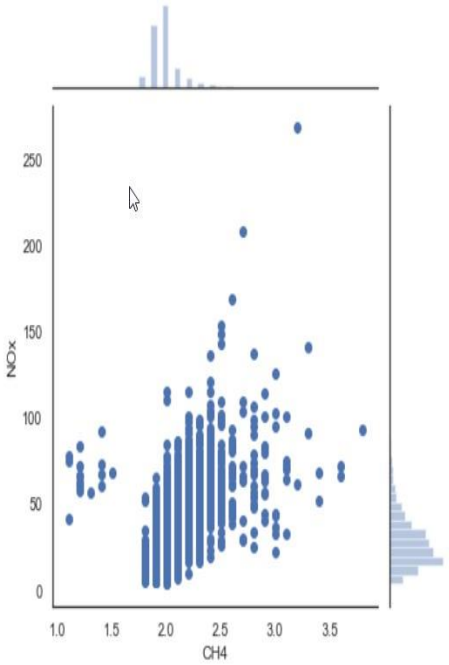


Fig 8.21 – jointplot3

```
In [45]: sns.set(style="white", color_codes=True)
         sns.jointplot(x='CO',y='SO2',data=data)

Out[45]: <seaborn.axisgrid.JointGrid at 0x2c8cca53400>
```

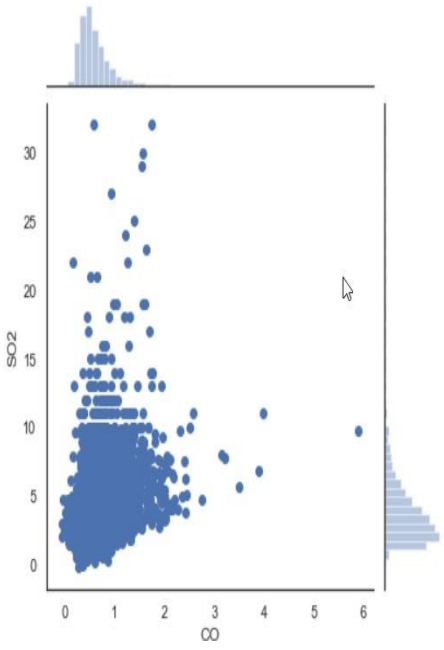


Fig 8.22 – jointplot4

```
In [46]: sns.set(style="white", color_codes=True)
         sns.jointplot(x='NO2',y='O3',data=data)

Out[46]: <seaborn.axisgrid.JointGrid at 0x2c8ccc09da0>
```

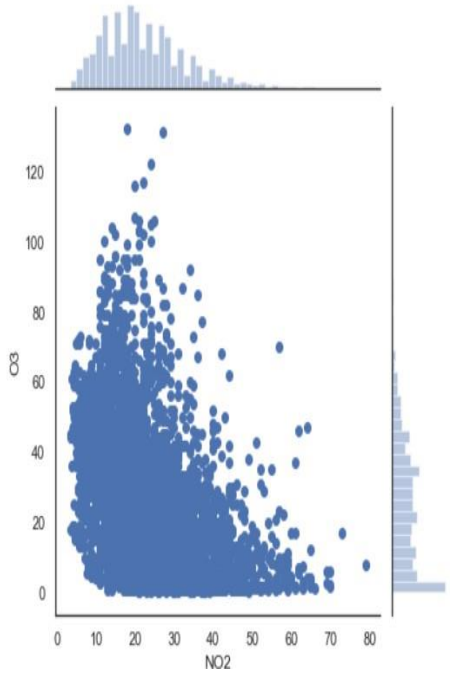


Fig 8.23 – jointplot5

```
In [47]: sns.set(style="white", color_codes=True)
         sns.jointplot(x='TEMP',y='RH',data=data)

Out[47]: <seaborn.axisgrid.JointGrid at 0x2c8cce1c9b0>
```

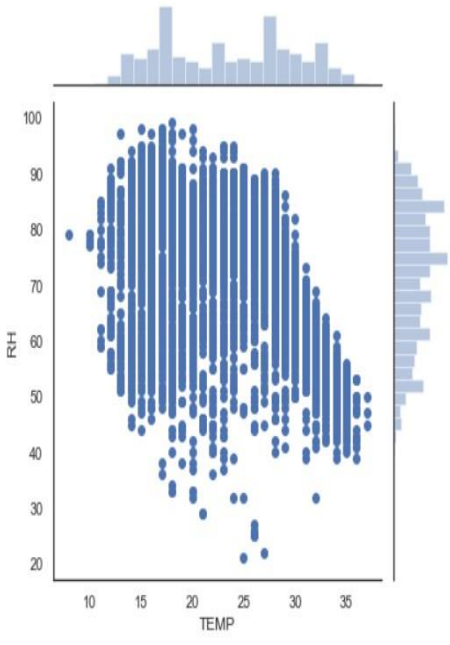


Fig 8.24 – jointplot6

```
In [48]: sns.set(style="white", color_codes=True)
         sns.jointplot(x='NO2',y='CO',data=data)
Out[48]: <seaborn.axisgrid.JointGrid at 0x2c8ccbf5860>
```

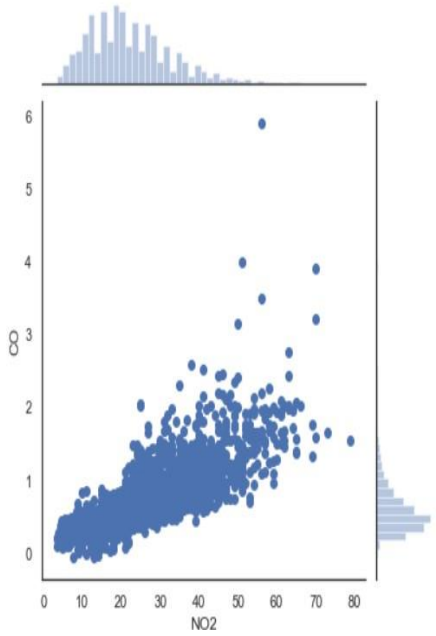


Fig 8.25 – jointplot7

Train Test Split

Now its time to split our data into a training set and a testing set!

```
In [49]: from sklearn.model_selection import train_test_split

In [50]: X = data.drop('label', axis=1)
         y = data['label']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

In [51]: X_train.to_excel('Traning_Testing/X_train.xlsx')
         X_test.to_excel('Traning_Testing/X_test.xlsx')
         y_train.to_excel('Traning_Testing/y_train.xlsx')
         y_test.to_excel('Traning_Testing/y_test.xlsx')
```

Fig 8.26 – train test split

Random Forest model

```
In [61]: from sklearn.ensemble import RandomForestClassifier

In [62]: rfc = RandomForestClassifier(n_estimators=600)

In [63]: rfc.fit(X_train,y_train)

Out[63]: 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=600, n_jobs=1,
                                oob_score=False, random_state=None, verbose=0,
                                warm_start=False)
```

Fig 8.27 – Random Forest Classifier

Predictions ¶

```
In [64]: predictions = rfc.predict(X_test)

In [65]: predictions

Out[65]: array([0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 2, 1, 0, 1, 1, 0, 0, 1, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
                0, 0, 2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
                0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 2,
                1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
                0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
                0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
                1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                0, 1, 1, 0, 0, 0, 1, 0, 2, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 2,
                0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
                0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
                1, 1, 1, 1, 2, 0, 0, 0, 1, 1, 0, 1, 0, 0, 2, 0, 1, 1, 0, 0, 1, 1,
                0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 2, 0, 0, 0,
```

Fig 8.28 - prediction

```
In [100]: #NAVIE BAYES ALGORITHM
          from sklearn.naive_bayes import GaussianNB
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import classification_report,confusion_matrix
          from sklearn.metrics import precision_recall_fscore_support

In [101]: model=GaussianNB()
          model=model.fit(X_train, y_train)

In [102]: predictNB=model.predict(X_test)
```

Fig 8.29 – Naïve Bayes Classifier

```
In [72]: #KNN ALGORITHM
         import itertools
         def plot_confusion_matrix(cm, classes,title='Confusion matrix'):

             plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=45)
             plt.yticks(tick_marks, classes)


             fmt = 'd'
             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, format(cm[i, j], fmt),
                          horizontalalignment="center",
                          color="white" if cm[i, j] > thresh else "black")

             plt.ylabel('Actual label')
             plt.xlabel('Predicted label')
             plt.tight_layout()
             plt.show()
```

Fig 8.30 – KNN Classifier

```
In [73]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import classification_report, accuracy_score
         from sklearn.neighbors import KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=1)

In [74]: knn.fit(X_train,y_train)

Out[74]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                    weights='uniform')

In [75]: pred = knn.predict(X_test)
         print(pred)

         [0 0 0 ... 0 0 1]
```

Fig 8.31 – predictions

```
In [82]: CO = X_test.CO
         SO2 = X_test.SO2
         NO2 = X_test.NO2
         O3 = X_test.O3
         PM10 = X_test.PM10

In [83]: CO_mean = CO.mean()
         SO2_mean = SO2.mean()
         NO2_mean = NO2.mean()
         O3_mean = O3.mean()
         PM10_mean = PM10.mean()
```

Fig 8.32 – pollutants data

```
In [84]: def checkCo(q):
             v = ''
             if q > 0 and q <= 1:
                 v = "Good"
             elif q > 1.1 and q <= 2.0:
                 v = "Satisfactory"
             elif q > 2.1 and q <= 10:
                 v = "Moderately Polluted"
             elif q > 10 and q <= 17:
                 v = "Poor"
             elif q > 17 and q <= 34:
                 v = "Very Poor"
             else:
                 v = "Severe"

             print('CO--> %2.3f    %s' %(q, v))
```

Fig 8.33 – CO analysis

```
In [85]: def checkSO2(q):
             v = ''
             if q > 0 and q <= 40:
                 v = "Good"
             elif q > 41 and q <= 80:
                 v = "Satisfactory"
             elif q > 81 and q <= 380:
                 v = "Moderately Polluted"
             elif q > 381 and q <= 800:
                 v = "Poor"
             elif q > 801 and q <= 1600:
                 v = "Very Poor"
             else:
                 v = "Severe"

             print('SO2--> %2.3f    %s' %(q, v))
```

Fig 8.34 – SO2 analysis

```
In [86]: def checkNO2(q):
             v = ''
             if q > 0 and q <= 40:
                 v = "Good"
             elif q > 41 and q <= 80:
                 v = "Satisfactory"
             elif q > 81 and q <= 180:
                 v = "Moderately Polluted"
             elif q > 181 and q <= 280:
                 v = "Poor"
             elif q > 281 and q <= 400:
                 v = "Very Poor"
             else:
                 v = "Severe"

             print('NO2--> %2.3f    %s' %(q, v))
```

Fig 8.35 – NO2 analysis

```
In [87]: def checkO3(q):
             v = ''
             if q > 0 and q <= 50:
                 v = "Good"
             elif q > 51 and q <= 100:
                 v = "Satisfactory"
             elif q > 101 and q <= 168:
                 v = "Moderately Polluted"
             elif q > 169 and q <= 208:
                 v = "Poor"
             elif q > 209 and q <= 748:
                 v = "Very Poor"
             else:
                 v = "Severe"

             print('O3--> %2.3f    %s' %(q, v))
```

Fig 8.36 – O3 analysis

```
In [88]: def checkPM(q):
             v = ''
             if q > 0 and q <= 50:
                 v = "Good"
             elif q > 51 and q <= 100:
                 v = "Satisfactory"
             elif q > 101 and q <= 250:
                 v = "Moderately Polluted"
             elif q > 251 and q <= 350:
                 v = "Poor"
             elif q > 351 and q <= 430:
                 v = "Very Poor"
             else:
                 v = "Severe"

             print('PM--> %2.3f   %s' %(q, v))
```

Fig 8.37 – PM2.5 analysis

CHAPTER 9
SYSTEM TESTING

9.1 Introduction

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

During the development of software, errors can be injected at any stage. However, requirements and design errors are likely to remain undetected. During testing, the program to be tested is executed with a set of test cases and output of program for test cases are evaluated to describe the program performance to expected level. No system design is perfect because of communication problem, programmer ‘s negligence or constraints create errors that must be eliminated before the system is ready for use of acceptance. There are various types of test. Each test type addresses a specific testing requirement. Following sections describes the system testing and test cases.

9.2 Unit Testing

This is the first level of testing, where different components are tested against the requirement specification for the individual components. Unit testing is essential for verification of the code produced during the coding phase and hence the goal is to test internal logic of those components.

9.3 System Testing

System testing is performed on the entire system in the context of functional requirements specification and system requirement specifications. System testing tests not only the design, but also the behaviour and the expectations. It is also to test up to and beyond the bounds defined in the software requirement specification. System testing for our platform is accomplished by the following test cases. These test cases show how each and every component interacts with each other as a dynamic system.

9.4 Level of Testing Used in Project

9.4.1 Unit testing

Initialization testing is the first level of dynamic testing and is first the responsibility of developers and then that of the test engineers. Unit testing is performed after the expected test results are met or differences are explainable/acceptable.

9.4.2 Integration testing

All module which make application are tested. Integration testing is to make sure that the interaction of two or more components produces results that satisfy functional requirement**.**

9.4.3 System testing

To test the complete system in terms of functionality and non-functionality. It is black box testing, performed by the Test Team, and at the start of the system testing the complete system is configured in a controlled environment.

9.4.4 Functional testing

The outgoing links from all the pages from specific domain under test. Test all internal links. Test links jumping on the same pages. Check for the default values of fields. Wrong inputs to the fields in the forms.

9.4.5 Alpha testing

Alpha testing is final testing before the software is released to the general public. This testing is conducted at the developer site and in a controlled environment by the end user of the software.

9.4.6 Beta testing

The beta test is conducted at one or more customer sites by the end user of the software. The beta test is conducted at one or more customer sites by the end user of the software.

CHAPTER 10

RESULTS & SCREENSHOTS

1. Random Forest Results:

Now create a classification report from the results

```
In [66]: from sklearn.metrics import classification_report,confusion_matrix

In [67]: print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	622
1	1.00	1.00	1.00	343
2	1.00	1.00	1.00	34
3	1.00	1.00	1.00	1
avg / total	1.00	1.00	1.00	1000

Fig 10.1 – classification report

```
In [68]: from sklearn.metrics import precision_recall_fscore_support

In [69]: cr = precision_recall_fscore_support(y_test,predictions,average='weighted')

In [70]: precision = cr[0]
         recall = cr[1]
         f1score = cr[2]

In [75]: li_x = ["", 'Precision', 'Recall', 'F1 Score']
         li_y = [0, float(precision), float(recall), float(f1score)]

In [76]: N = len(li_x)

         ind = np.arange(N)

         fig = plt.figure(figsize=(10,5), dpi=80)
         plt.bar(ind, li_y)
         plt.xticks(ind, li_x)
         plt.title("Classification Report")
         plt.grid(True)
         plt.show()
```

Fig 10.2 – evaluation metrics

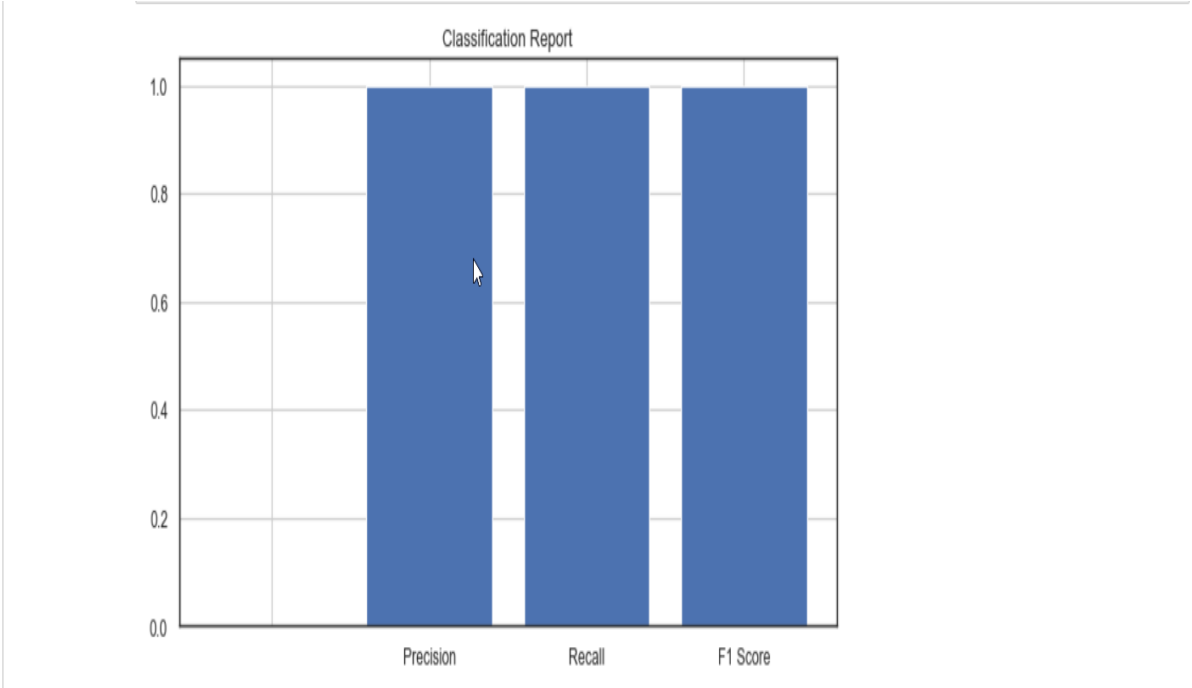


Fig 10.3 – classification report

```
In [77]: skplt.metrics.plot_confusion_matrix(y_test, predictions, normalize=False)
         plt.show()
```

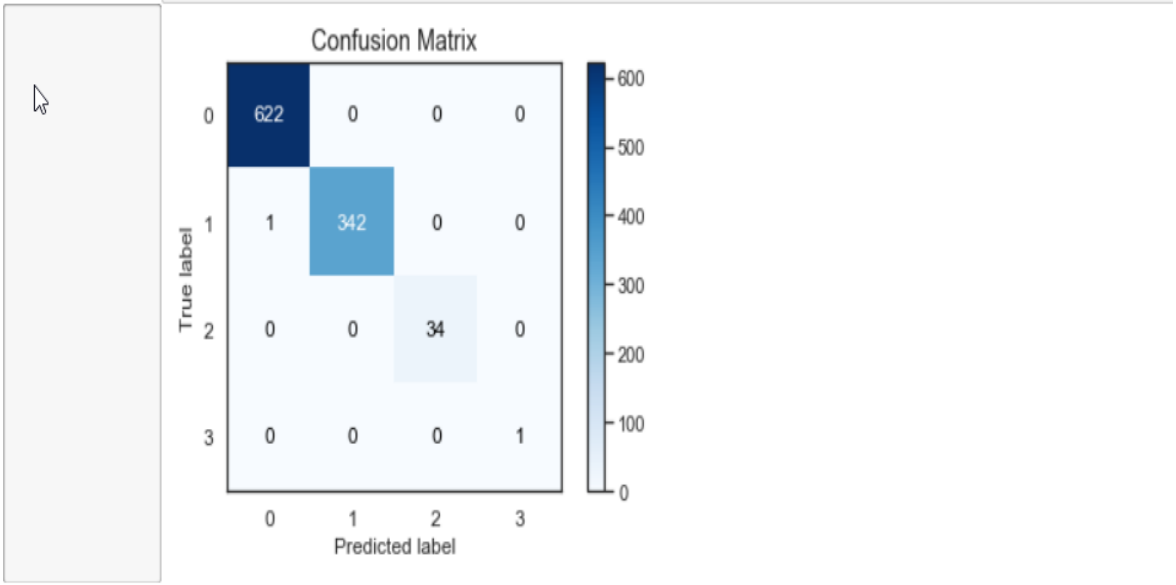


Fig 10.4 – confusion matrix

2. Naïve Bayes Results:

```
In [103]: crDT = precision_recall_fscore_support(y_test,predictNB,average='weighted')
          precisionNB = crDT[0]
          recallNB = crDT[1]
          f1scoreNB = crDT[2]
          precisionNB

Out[103]: 0.8423718726829269

In [104]: classificationReportNB = classification_report(y_test,predictNB)

In [105]: confusionmatrixNB=confusion_matrix(y_test,predictNB)
```

Fig 10.5 – precision

```
In [106]: print(classificationReportNB)
                  precision    recall  f1-score   support

               0       0.88      0.92      0.90     29727
               1       0.78      0.65      0.71     12606
               2       0.69      0.79      0.74      1200
               3       0.08      0.34      0.13       107
               4       0.84      0.78      0.81        88

       micro avg       0.84      0.84      0.84     43728
       macro avg       0.65      0.70      0.66     43728
    weighted avg       0.84      0.84      0.84     43728
```

Fig 10.6 – classification report

```
In [107]: print(confusionmatrixNB)
[[27462  2093     2   160    10]
 [ 3823  8219   351   212     1]
 [    0   230   948    22     0]
 [    0     1    68    36     2]
 [    0     0     0    19    69]]
```

Fig 10.7 – confusion matrix

```
In [108]: skplt.metrics.plot_confusion_matrix(y_test, predictNB, normalize=False)
          plt.show()
```

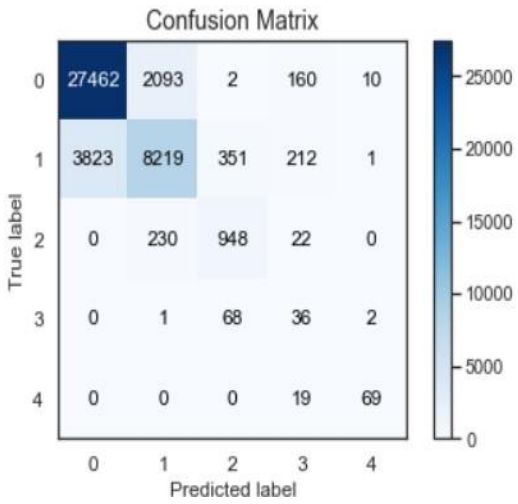


Fig 10.8 – confusion matrix plot

3. KNN Results:

```
In [76]: acc1 = accuracy_score(y_true=y_test, y_pred= pred)
         print(acc1)

         0.9692188071716062

In [77]: from sklearn.metrics import classification_report,confusion_matrix
         from sklearn.metrics import precision_recall_fscore_support

In [78]: cr = precision_recall_fscore_support(y_test,pred,average='weighted')

In [79]: precisionKNN = cr[0]
         recall = cr[1]
         f1score = cr[2]

In [80]: cm1 = confusion_matrix(y_true=y_test,y_pred=pred)
```

Fig 10.9 – accuracy

```
In [81]: print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     29727
           1       0.95      0.94      0.95     12606
           2       0.91      0.92      0.92      1200
           3       0.94      0.88      0.91       107
           4       0.96      0.99      0.97        88

   micro avg       0.97      0.97      0.97     43728
   macro avg       0.95      0.94      0.95     43728
weighted avg       0.97      0.97      0.97     43728
```

```
In [82]: print(cm1)
```

```
[[29186   541     0     0     0]
 [  601 11908    97     0     0]
 [    0    88  1107     5     0]
 [    0     0     9    94     4]
 [    0     0     0     1    87]]
```

Fig 10.10 – evaluation metrics

```
In [83]: level=data['label'].unique()
         print(level)
```

```
[3 2 1 0 4]
```

```
In [84]: plot_confusion_matrix(cm1, level,title='Confusion matrix')
```

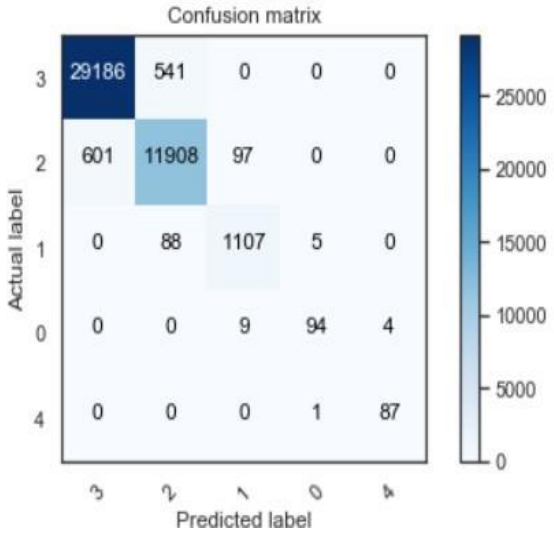


Fig 10.11 – confusion matrix plot

```
In [124]: #COMPARING RESULTS OF NAIVE BAYES AND RANDOM FOREST AND KNN
          li_x = ['NB', 'RF','KNN']
          li_y = [precisionNB, precision, precisionKNN]

In [125]: print(li_y)
          sns.barplot(x=li_x, y=li_y)

          [0.8423718726829269, 0.9995850995384271, 0.9691867146168233]

Out[125]: <matplotlib.axes._subplots.AxesSubplot at 0x171cc6c3320>
```

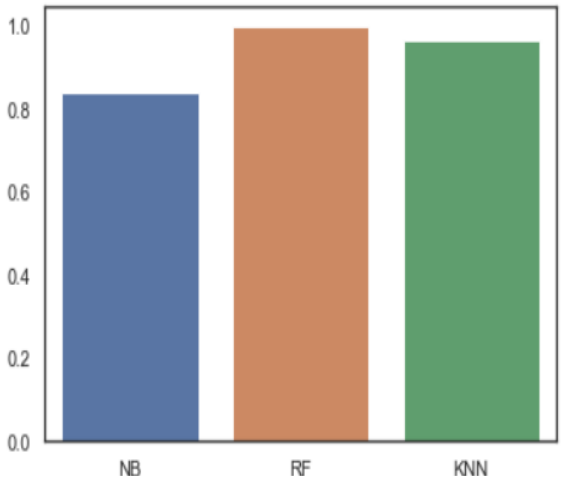


Fig 10.12 – comparing results

4. Feature Analysis:

Fig 10.13 – Analysis

AQI Category, Pollutants and Health Breakpoints

AQI Category (Range)	PM_{10} (24hr)	$PM_{2.5}$ (24hr)	NO_2 (24hr)	O_3 (8hr)	CO (8hr)	SO_2 (24hr)	NH_3 (24hr)	Pb (24hr)
Good (0-50)	0-50	0-30	0-40	0-50	0-1.0	0-40	0-200	0-0.5
Satisfactory (51-100)	51-100	31-60	41-80	51-100	1.1-2.0	41-80	201-400	0.5-1.0
Moderately polluted (101-200)	101-250	61-90	81-180	101-168	2.1-10	81-380	401-800	1.1-2.0
Poor (201-300)	251-350	91-120	181-280	169-208	10-17	381-800	801-1200	2.1-3.0
Very poor (301-400)	351-430	121-250	281-400	209-748	17-34	801-1600	1200-1800	3.1-3.5
Severe (401-500)	430+	250+	400+	748+	34+	1600+	1800+	3.5+

AQI	Associated Health Impacts
Good (0-50)	Minimal impact
Satisfactory (51-100)	May cause minor breathing discomfort to sensitive people.
Moderately polluted (101–200)	May cause breathing discomfort to people with lung disease such as asthma, and discomfort to people with heart disease, children and older adults.
Poor (201-300)	May cause breathing discomfort to people on prolonged exposure, and discomfort to people with heart disease.
Very poor (301-400)	May cause respiratory illness to the people on prolonged exposure. Effect may be more pronounced in people with lung and heart diseases.
Severe (401-500)	May cause respiratory impact even on healthy people, and serious health impacts on people with lung/heart disease. The health impacts may be experienced even during light physical activity.

Fig 10.14 - pollutants impact

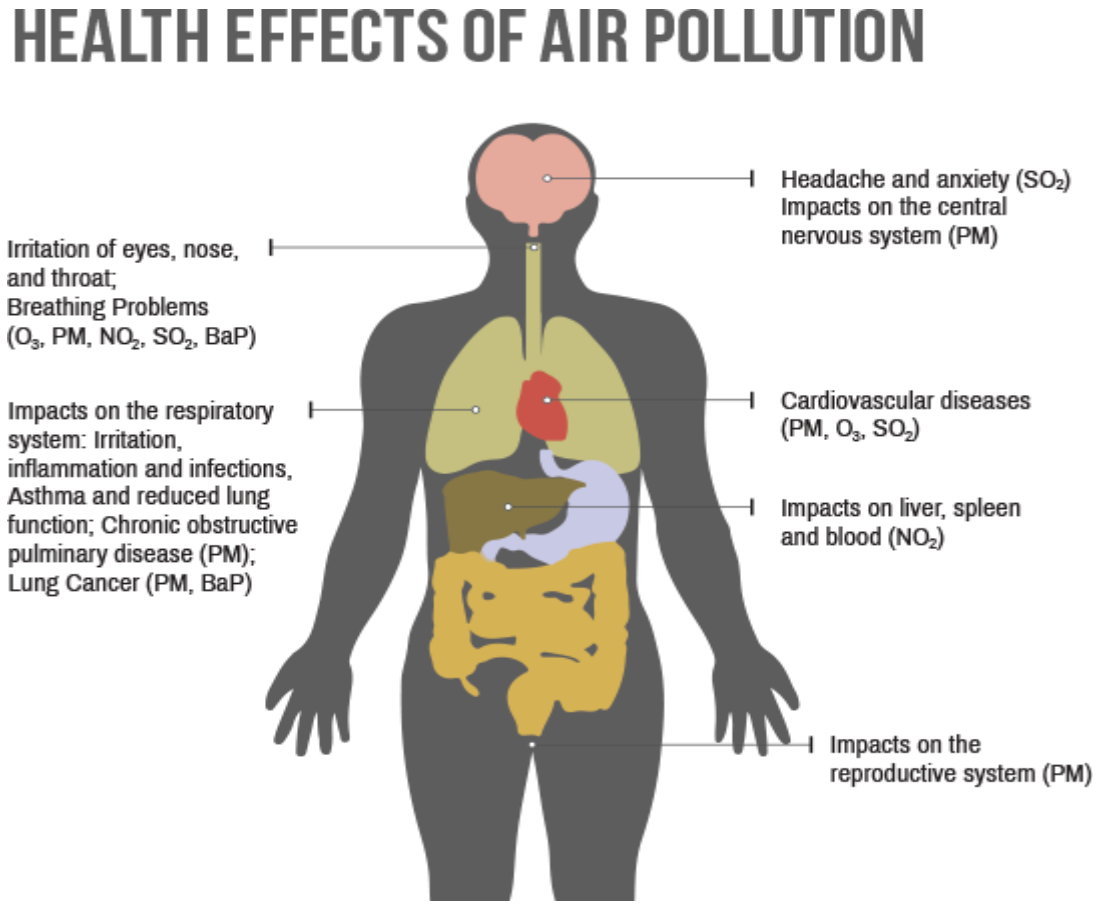


Fig 10.15 – health effects

CHAPTER 11
CONCLUSION & SCOPE

The three important topics in the area of urban air computing are: interpolation, prediction, and feature analysis of fine-gained air quality. The solutions to these topics can provide crucial information to support air pollution control, and consequently generate great societal and technical impacts. Most existing efforts focus on solving the three problems separately by establishing different models. In this paper, we develop a general and effective approach called DAL to unify the interpolation, prediction, feature selection and analysis of the fine-grained air quality into one model. Combining with feature selection, association analysis discovers the importance of different input features to the predictions of the neural networks. The proposed feature selection and analysis method has the ability to reveal some inner mechanism of the blackbox deep network models.

This analysis plays very important role to create an alarm in the smart city regarding smog and to devise environmental policies. In this project with the public data of metropolitan city, our model predicts the density of the gases that causes smog (air pollution) in the regions that are based on data combined (across the years) and largely clean version of the Historical Daily Ambient Air Quality Data released by the Ministry of Environment and Forests and Central Pollution Control Board of India. All of these data are directly or indirectly available over the internet. Also, the algorithms can be applied for other cities.

Future Scope: We can implement UI based user friendly system with the aid of Django framework for live AQI tracing of the cities.

CHAPTER 12

REFERENCES & WEBLINKS

12.1 References

[1]. “Spatiotemporal Interpolation Methods for Air Pollution Exposure”, Lixin Li, Xingyou Zhang and James B. Holt et al.

[2]. “U-Air: When Urban Air Quality Inference Meets Big Data”, Yu Zheng, Furui Liu, Hsun-Ping Hsieh.

[3]. “Inferring Air Quality for Station Location Recommendation Based on Urban Big Data”, Hsun-Ping Hsieh, Shou-De Lin , Yu Zheng.

[4]. “Model for Forecasting Expressway Fine Particulate Matter and Carbon Monoxide Concentration: Application of Regression and Neural Network Models”, Salimol Thomas & Robert B Jacko.

[5]. “Forecasting Fine-Grained Air Quality Based on Big Data”, Yu Zheng, Xiuwen Yi, Ming Li et al.

[6]. “Graph-Based Semi-Supervised Learning with Multi-Label”, Zheng-Jun Zha, Tao Mei et al.

[7]. “Combining Labelled and Unlabelled Data with Co-Training”, Avrim Blum, Tom Mitchell.

[8]. “Co-training for Predicting Emotions with Spoken Dialogue Data”, Beatriz Maeireizo, Diane Litman and Rebecca Hwa.

[9]. “Learning with Limited and Noisy Tagging”, Yingming Liy, Zhongang Qiy, Zhongfei (Mark) Zhang et al.

[10]. “Semi-Supervised Recursive Auto encoders for Predicting Sentimen Distributions”, Richard Socher Jeffrey Pennington et al.

12.2 Weblinks

- https://data.gov.in/
- https://www.researchgate.net/publication/320867332_Deep_Air_Learning_Interpolation_Prediction_and_Feature_Analysis_of_Fine-grained_Air_Quality.
- https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
- https://towardsdatascience.com/random-forest-classification-and-its-implementation-d5d840dbead0
- https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761