

## **DECLARATION**

We, hereby declare that the Synopsis entitled “**PREDICTION OF NOVEL PROTEINS IN THE HUMAN-DENGUE INTERACTION PATHWAY USING MACHINE LEARNING TECHNIQUES**” submitted to the Department of Computer Science & Engineering, NSEC, is an original work done by us under the guidance of Prof. Piyali Chatterjee, Professor, CSE, NSEC and this work is submitted in partial fulfillment of the requirement for the award of bachelor’s degree in computer science and engineering. It has not been submitted to any other university or Institute for the award of any degree.

**Signature**

*Raunak Saha Foudes*

**Signature**

*Rudra Narayan Roy*

# **CERTIFICATE**

This is to certify that the Project entitled “**PREDICTION OF NOVEL PROTEINS IN THE HUMAN-DENGUE INTERACTION PATHWAY USING MACHINE LEARNING TECHNIQUES**” submitted by RAUNAK SAHA FOUZDER (10900120170), RUDRA NARAYAN ROY (10900120013) is a record of Bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of **Bachelor of Technology (Computer Science & Engineering)** at **Netaji Subhash Engineering College, Garia**. This work is done during the year **2023-2024**, under my guidance.

Date: 22/05/2024

---

Prof. (Dr.) Piyali Chatterjee

HEAD,  
Department of Computer Science & Engg.

NETAJI SUBHASH ENGINEERING COLLEGE  
TECHNOCITY, GARIA, KOLKATA - 700152

---

Prof. (Dr.) Piyali Chatterjee

Professor,  
Department of Computer Science & Engg.

NETAJI SUBHASH ENGINEERING COLLEGE  
TECHNOCITY, GARIA, KOLKATA - 700152

## **EXAMINER'S APPROVAL**

We hereby approve this dissertation titled “**PREDICTION OF NOVEL PROTEINS IN THE HUMAN-DENGUE INTERACTION PATHWAY USING MACHINE LEARNING TECHNIQUES**” carried out by:  
RAUNAK SAHA FOUZDER, Roll no: 10900120170, Regd. No. 201090100110003 of 2020-21  
RUDRA NARAYAN ROY, Roll no: 10900120013, Regd. No. 201090100110160 of 2020-21  
under the guidance of Prof. (DR.) Piyali Chatterjee of Netaji Subhash Engineering College,  
Kolkata in partial fulfillment of requirements for award of the degree Bachelor of Technology  
(B. Tech) in Computer Science & Engineering of Maulana Abul Kalam Azad University of  
Technology.

Date: 22.05.2024

Examiner's Signatures:

1. ....
2. ....
3. ....

## **ACKNOWLEDGEMENTS**

We are profoundly grateful to our mentors, Dr. Piyali Chatterjee and Prof. Sovan Saha, for their expert guidance and continuous encouragement throughout this project. Their support has been instrumental in ensuring its successful completion. We would also like to thank our friend, Pubali Basak, for her valuable help .....

**RAUNAK SAHA FOUZDER**  
**RUDRA NARAYAN ROY**

# ABSTRACT

It is important to study the human proteins that get involved in the dengue-human protein interaction pathway so that appropriate medications can be researched to somehow disrupt that pathway or block the dengue proteins from interacting with the host. In this paper, various machine learning algorithms, such as logistic regression, random forests, support vector machines, extra trees, Naive Bayes, Adaboost, XGBoost etc. are employed to identify patterns and relationships within the data related to the defining characteristics of the human proteins already known and confirmed to get involved in the pathway so that we can predict novel proteins that may play a role in the pathway and hence provide a direction for further research increasing the efficiency. Identification of novel proteins will help point out new targets for therapeutic drugs which in turn will help make the lives of patients easier and will significantly decrease the mortality rate. The use of diverse algorithms and comprehensive data allows us to construct robust predictive models. Machine learning techniques hold great promise, offering researchers valuable tools to enhance their breakthroughs. The findings presented in this study underscore the potential of machine learning in addressing one of the most significant global health challenges of our time.

**KEYWORDS:** Human-Dengue Protein Interaction network, Machine Learning, Drug repurposing;

## **CONTENTS**

<b>CHAPTERS</b>	<b>PAGE NO</b>
<b>1.INTRODUCTION</b>	<b>7</b>
<b>2.LITERATURE REVIEW</b>	<b>9</b>
<b>3.THE PROPOSED WORKING MODEL</b>	<b>15</b>
<b>4.DETAILED EXPERIMENTATION</b>	<b>17</b>
<b>5.RESULTS AND DISCUSSION</b>	<b>61</b>
<b>6.CONCLUSION</b>	<b>75</b>
<b>7. REFERENCES</b>	<b>76</b>
<b>8. APPENDIX</b>	<b>78</b>

# **CHAPTER 1**

## **INTRODUCTION**

Dengue fever, caused by the dengue virus (DENV), remains a significant global health concern, particularly in tropical and subtropical regions. Despite decades of research, effective therapeutics and vaccines against dengue fever are still lacking, highlighting the urgent need for innovative approaches to combat this debilitating disease. Understanding the intricate interplay between viral and host proteins is crucial for developing targeted interventions. Recent advancements in machine learning (ML) algorithms offer a promising avenue to decipher complex biological interactions and identify novel human proteins involved in the dengue-human protein interaction pathway.

The dengue virus employs sophisticated strategies to hijack host cellular machinery for its replication and propagation. Central to these processes are the interactions between viral and host proteins, which orchestrate various stages of the viral life cycle and modulate the host immune response. Traditional experimental methods for elucidating protein-protein interactions (PPIs) are labor-intensive, time-consuming, and often limited by their capacity to explore the vast complexity of the human proteome. In contrast, ML algorithms provide a powerful tool to analyze large-scale biological datasets and predict novel protein interactions with high accuracy and efficiency.

This research aims to leverage ML techniques to identify previously unrecognized human proteins that may participate in the dengue-human protein interaction network. We construct comprehensive computational models to uncover hidden patterns and associations within complex biological systems. Through feature selection, dimensionality reduction, and model

optimization, ML algorithms can effectively prioritize candidate proteins with the highest likelihood of involvement in the dengue-human protein interaction pathway.

The discovery of novel human proteins implicated in dengue infection holds immense potential for advancing our understanding of disease pathogenesis and facilitating the development of innovative therapeutic strategies. Firstly, identifying host factors critical for viral replication and propagation can unveil new targets for antiviral drug development. By disrupting essential protein interactions or modulating host pathways necessary for viral survival, novel therapeutics can inhibit viral replication and mitigate disease progression. Furthermore, elucidating the molecular mechanisms underlying dengue-induced immunopathology can inform the design of immunomodulatory agents to alleviate severe disease manifestations.

Moreover, the integration of ML-driven predictions with experimental validation approaches, such as biochemical assays and functional genomics studies, can validate the biological relevance of candidate proteins and refine our understanding of their roles in dengue pathogenesis.

Furthermore, the discovery of novel human proteins implicated in dengue infection can also have broader implications for our understanding of host-virus interactions across different viral pathogens. Many viruses, including Zika virus, chikungunya virus, and influenza virus, share common strategies with dengue virus in manipulating host cellular processes for their replication and dissemination. Therefore, insights gained from studying the dengue-human protein interaction pathway may have far-reaching implications for combating a wide range of viral infections.



## **CHAPTER 2**

# **LITERATURE REVIEW**

Several studies have been conducted to understand the complex interactions between dengue virus (DENV) and its hosts, *Homo sapiens* and the *Aedes aegypti* mosquito vector, with implications for both understanding the virus's life cycle and potential therapeutic interventions. Sessions et al. identified insect host factors crucial for DENV propagation, highlighting potential avenues for controlling infection in both vectors and mammalian hosts. Doolittle and Gomez employed a computational approach to predict thousands of interactions between DENV and human proteins, offering hypotheses for further experimental investigation and therapeutic targeting. Le Sommer et al. discovered potential anti-flaviviral drug targets, emphasizing the inhibition of GRK2 function as a novel approach. Mairiang et al. identified numerous host proteins interacting with DENV proteins across serotypes, advancing towards a comprehensive host-dengue protein interactome. Savidis et al. and Viktorovskaya et al. both contributed to understanding the host factors crucial for flavivirus replication, identifying key proteins involved in endocytosis, transmembrane protein processing, and RNA binding, providing insights into therapeutic strategies. Rothan and Kumar highlighted the importance of ER-associated complexes in flavivirus replication, suggesting them as targets for broad-spectrum anti-flavivirus drugs. Farooq et al. constructed a comprehensive protein-protein interaction map between DENV and human proteins, offering further insights into the virus-host interaction landscape. Together, these studies contribute to elucidating the molecular mechanisms of DENV infection and offer potential targets for therapeutic intervention. Further details about individual papers are given below: -

[1] (Sessions et al., 2009a) *M. Sessions et al.* declares that given their compact genomes, dengue viruses (DENV-1–4) and other flaviviruses probably require an extensive number of host factors; however, only a limited number of human host factors have been identified. So here they identify insect host factors required for DENV-2 propagation in *Drosophila melanogaster* and identified 116 candidate dengue virus host factors (DVHF). The dipteran DVHFs had 82 readily recognizable human homologues and 42 of these are human DVHFs. This work suggests new approaches to control infection in the insect vector and the mammalian host. Results: - Some Human proteins found interacting in the pathway include: - EPS15, EPS15L1, NPR2, NPRL2, SEC61B, EXD2, CNOT2.

[2] (Doolittle & Gomez, 2011a) Janet M. Doolittle, Shawn M. Gomez proposed about implementing a computational approach to predict interactions between Dengue virus (DENV) and both of its hosts, *Homo sapiens* and the insect vector *Aedes aegypti* with their approach based on structural similarity between DENV and host proteins. They predicted over 4,000 interactions between DENV and humans, as well as 176 interactions between DENV and *A. aegypti*. It concludes by stating that dengue virus manipulates cellular processes to its advantage through specific interactions with the host's protein interaction network. The interaction networks presented in the paper provide a set of hypotheses for further experimental investigation into the DENV life cycle as well as potential therapeutic targets. Results: - Some Human proteins found interacting in the pathway include: HSP90AA1, HSPA4, HSPA5, CALR, CD14 etc.

[3] (Le Sommer et al., 2012) *Caroline Le Sommer and her co-authors* focus on the discovery of host factors that regulate the fate of flaviviruses in infected cells could provide insight into the molecular mechanisms of infection and therefore facilitate the development of anti-

flaviviral drugs. They performed an extensive screening of the human host proteins and then tried to find commonalities between flaviviruses. These candidates were compared to host factors previously identified for West Nile virus (WNV) and dengue virus (DENV) which pointed out a potential requirement for the G protein-coupled receptor kinase family, GRKs, for flaviviral infection. Their findings suggest that inhibition of GRK2 function may constitute a new approach for treatment of flavivirus associated diseases. Results: - Some Human proteins (G-proteins) found interacting in the pathway include: - GRK2, GRK4, GRK6

[4] (Mairiang et al., 2013) *Dumrong Mairiang and his co-authors*. have used high throughput yeast two-hybrid screening to identify mosquito and human proteins that physically interact with dengue proteins. They have tested and identified host protein against the proteins from all four serotypes of dengue to identify interactions that are conserved across serotypes and further confirmed many of the interactions using co-affinity purification assays. They also ended up identifying some previously detected interactions and many new ones, moving us closer to a complete host – dengue protein interactome. Results: - Some Human proteins found interacting in the pathway include: - GOLGB1, RPL24, GTPBP4 etc. For the complete list visit [here](#)

[5] (Savidis et al., 2016) *George Savidis and his co-authors* try to identify DENV and ZIKV dependencies using orthologous RNAi and CRISPR/Cas9 screening approaches. RNAi and CRISPR/Cas9 screens were used to find flavivirus dependencies. The screens recovered the ZIKV entry factor AXL as well as multiple host factors involved in endocytosis (RAB5C and RABGEF), heparin sulfation (NDST1 and EXT1), and transmembrane protein processing and maturation, including the endoplasmic reticulum membrane complex (EMC). Multiple host factors involved in endocytosis and transmembrane protein processing, including the endoplasmic reticulum membrane complex (EMC), are important for flaviviral replication.

Together, their studies generate a systems-wide view of human-flavivirus interactions. Results:

- Some Human proteins found interacting in the pathway include: - RTN1, RTN2, RTN3, DDX56, KDELR, RAB8A, RAB8B etc.

[6] (Viktorovskaya et al., 2016) *Viktorovskaya and team state that host factors that are required for viral amplification provide an attractive target for antiviral therapeutics for rapidly evolving RNA viruses. However, only a handful of DENV host factors are known and this study reports identification of 79 novel dengue RNA binding proteins using a large-scale analysis of cellular proteins that interact with the DENV RNA during a live infection in human cells. The method for identification of host factors described here is robust and broadly applicable to all RNA viruses, providing an avenue to determine the conserved or distinct mechanisms through which diverse viruses manage the viral RNA within cells. This study significantly increases the number of cellular factors known to interact with DENV and reveals how DENV modulates and usurps cellular proteins for efficient amplification. Results: - Some DENV Host factors found: - polypyrimidine tract-binding protein 1 (PTBP1), interleukin enhancer-binding factor 3 (ILF3), calreticulin (CALR), calnexin and heterogeneous nuclear ribonucleoproteins hnRNP H1 and hnRNP K, as well as a known DENV anti-viral protein—eukaryotic initiation factor 4A (eIF4A)—and 12 other proteins previously shown to associate with DENV RNA or proteins. For the entire list visit [here](#)*

[7] (Rothan & Kumar, 2019) *Hussin A. Rothan and Mukesh Kumar explore the aspect of Flavivirus replication in host cells that requires the formation of replication and assembly complexes on the cytoplasmic side of the endoplasmic reticulum (ER) membrane. This study highlights the fact that the ER multiprotein complexes are crucial for the formation of flavivirus replication and assembly complexes, and the ER complexes could be considered as a target for*

developing successful broad-spectrum anti-flavivirus drugs. They have concluded that Flaviviruses exploit the ER function during infection to gain optimal replication. Multiple independent genome-wide screen studies have identified several ER-associated complexes and individual proteins that are important for flavivirus replication. These complexes such as DNAJC14, Hrd1, EMC, and RTN 3.1A play a crucial role in the construction and function of virus replication and assembly complexes and hence can prove as promising host targets for developing broad-spectrum anti-flavivirus drugs. Results: - ER proteins Required for Flavivirus Replication and Assembly are: - DNAJC14, Hrd1 complex, Oligosaccharyltransferase (OST) complex, Reticulon 3 (RTN 3.1A) etc.

[8] (Farooq et al., 2023) *Qurrat ul Ain Farooq et al.*, constructed a comprehensive PPI map of DENV with its host Homo sapiens and performed various bioinformatics analyses which lead to the finding of 1195 interactions between 858 human and 10 DENV proteins out of which they considered top 5 human proteins with the maximum number of interactions with dengue viral proteins. The non-structural protein NS1 in DENV had the maximum number of interactions with the host protein, followed by NS5 and NS3. Among the human proteins, HBA1 and UBE2I were associated with 7 viral proteins, and 3 human proteins (CSNK2A1, RRP12, and HSP90AB1) were found to interact with 6 viral proteins. The DENV-NS1 protein was prioritized as a druggable protein based on the highest number of interactions with the human host. Results: - Some Human proteins found interacting in the pathway include: - ALDOB, APEH, APOH, CYP2C8 etc. For the list of all proteins visit

[here](#)

Our aim through this literature survey is to compile the list of human proteins participating in the protein-protein interaction pathway of Dengue and Human and use machine learning models to train and test on the data to verify the accuracy of the model. The 8 best performing models are taken to vote in the database of all the human proteins (except the ones used to train and test) to find prospective novel proteins that might get involved in the pathway to form a more curated list for researchers to focus their studies and resources on. This would increase the efficiency in which experiments are conducted for verification thereby resulting a deeper understanding of the molecular machinery and formulating relevant drugs to counter this deadly disease.

## CHAPTER 3

# THE PROPOSED WORKING MODEL

### Computational Steps of the Proposed Method:

**Step 1:** - Conducting Literature Survey and curating a list of relevant research papers which are relevant to our project proposal. This provides the necessary starting point and data for our project. The list of proteins is compiled from various sources including these research papers

**Step 2:** - After the list of proteins is obtained, we map Uniprot IDs [9] to the protein names. Web scraping is used to calculate the numerical values for the protein features using the corresponding Uniprot IDs [9] on Pfeature [10] and storing them in an excel sheet. Then a binary classified dataset is prepared using the data from literature review and proteins from the human proteins database from Uniprot [9]. We make sure that the dataset is balanced.

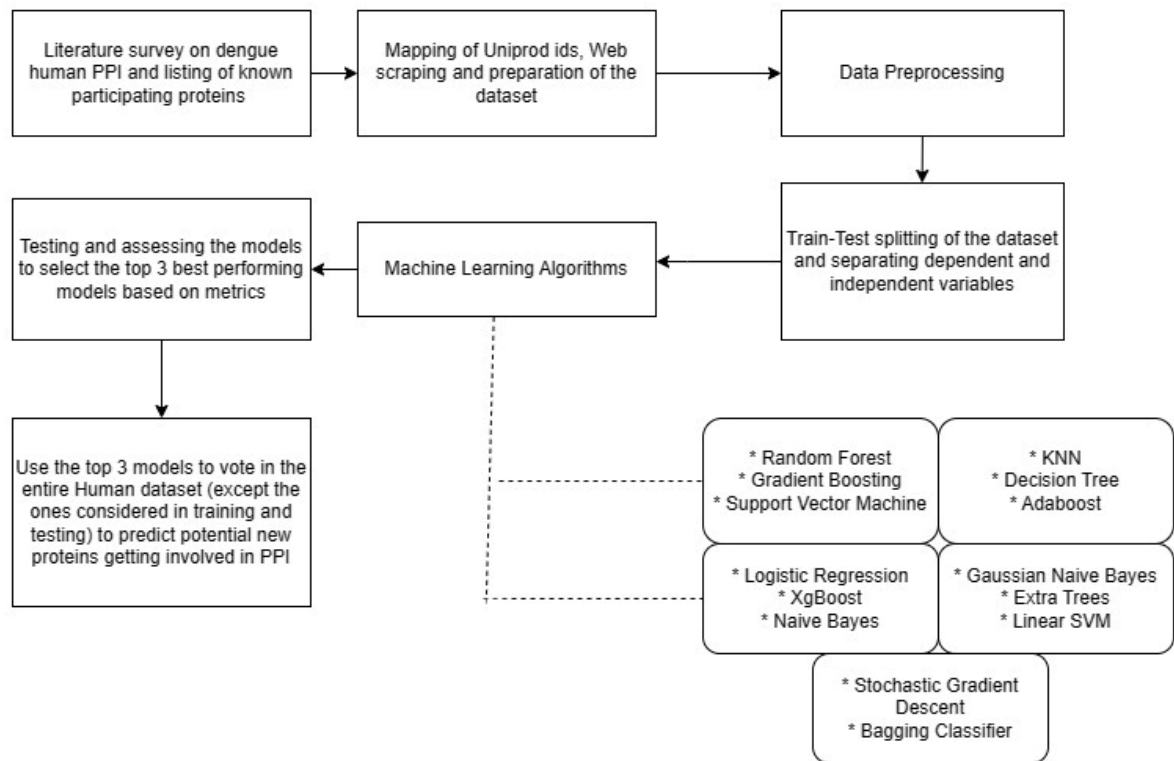
**Step 3:** - Data cleaning and data preprocessing is carried out, like removal of duplicates, clearing dataset of NaN values and normalizing the dataset. Unnecessary columns are dropped.

**Step 4:** - The independent and dependent variables are separated and the dataset is split into train and test parts in the ratio of 8:2

**Step 5:** - Various Machine Learning models are trained on the dataset and then they are evaluated based on the metrics regarding their performance on the test data and the top 8 performing models are selected.

**Step 6:** - Use the top 8 models to vote for individual human proteins that were not used to train or test the machine learning models and create a list. That list contains potential novel proteins that need to be considered for further study.

Following figure depicts the working of the proposed method.



**Figure 1: OVERALL ARCHITECTURE OF THE PROPOSED METHOD**



## **CHAPTER 4**

# **DETAILED EXPERIMENTATION**

### **WEB SCRAPING**

Web scraping is a method of extracting data from websites automatically, utilizing bots or web crawlers to navigate through web pages, collect information, and store it for analysis. It's a powerful tool employed across numerous domains, from business intelligence to academic research and beyond. By parsing HTML and other web page structures, scraping tools can retrieve specific data points.

However, ethical considerations are paramount in web scraping. While scraping publicly available data is generally permissible, scraping large amounts of data or violating a website's terms of service can lead to legal issues. Respect for a site's robots.txt file and rate-limiting requests are common practices to maintain ethical scraping behavior. Additionally, data privacy concerns arise when scraping personal information or copyrighted content.

Despite these challenges, web scraping offers immense benefits. It streamlines data collection processes, enabling researchers to gather vast amounts of data for analysis. With the right tools and ethical approach, web scraping is a valuable technique for extracting insights from the vast landscape of the internet.

In our research we have used it to scrape public data from Pfeature [10] for listing out the numerical values for the features of the proteins considered in the study using selenium module in Python. We create a Firefox driver and use it to crawl the website by automatically entering the Uniprot IDs [9] from the excel sheet and waiting for the website to load the results relating to the various Physico-chemical features of the corresponding protein and then the bot reads

the information and inserts it back into the same excel file besides the corresponding protein entry.

The list of proteins to be considered is a curated list obtained from compiling multiple research papers through literature survey and the entire sheet of all the known human proteins which will be used to find novel proteins getting involved in the dengue-human PPI has been obtained from Uniprot [9].

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
import pandas as pd

result_data = pd.read_csv("test_proteins_19k.csv")
column_list = ["PCP_PC", "PCP_NC", "PCP_NE", "PCP_PO", "PCP_NP", "PCP_AL", "PCP_CV", "PCP_AR", "PCP_AC"]
for idx in column_list:
    result_data.insert(len(result_data.columns), idx, None)

driver = webdriver.Firefox()
driver.get("https://webs.iiitd.edu.in/raghava/pfeature/physio.php")

# checking negatively charged
elem = driver.find_element(By.CSS_SELECTOR, "main.hoc > div:nth-child(1) > table:nth-child(2) > tbody:nt
elem.click()
```

FIGURE 2: WEB SCRAPING (CODE)

## THE DATASET

Data serves as the cornerstone for training machine learning models, providing the necessary input that enables algorithms to learn patterns, make predictions, and derive insights. Machine learning models generalize from data, allowing them to make predictions on new, unseen data. A rich and representative dataset enhances a model's ability to adapt and perform well in real-world scenarios.

Here we have created and used the dataset. The relevant information is compiled from trustable sources and has been organised in a structural format. There are 34 features that has been

organised in the dataset. 30 features relating to the physico-chemical properties of the proteins are present which are numerical in nature, two states the name of the protein and their corresponding Uniprot IDs [9], one states the index and the last one is the target variable. The dataset has 603 entries and we have made sure that the dataset is fairly balanced. There are 303 positive proteins (that are verified to play a role in the dengue-human PPI) and 300 negative human proteins (proteins that don't play a role in the dengue-human PPI) and have created a binary classification dataset. Positive proteins are denoted as 1 and negatives as 0 in the column named affects (target column). The features in details are stated below: -

Table 1: - Description of Attributes

Content Name	Description
Unnamed: 0	Provides index for the entries
Uniprot ID	The Uniprot ID [9] in human proteins serves as a unique identifier, facilitating the retrieval of comprehensive information about a specific protein, including its sequence, function, subcellular localization, and associated pathways, enabling researchers to delve into its biological roles and significance in health and disease.
Entry Name	The Entry Name in human proteins provides a concise identifier for a specific protein, aiding researchers in quickly accessing relevant information such as protein function, structure, and associated pathways, streamlining the investigation into its role in various biological processes and disease states.
Affects	The "Affects" feature in human proteins highlights the diverse biological processes, pathways, and cellular functions influenced by a specific protein, offering valuable insights into its role in health, disease, and therapeutic interventions.
Positively Charged (PCP_PC)	Positively charged (PCP_PC) residues in human proteins play crucial roles in various biological processes, such as protein-protein interactions, enzymatic activities, and cellular signaling pathways.
Negatively Charged (PCP_NC)	Negatively charged (PCP_NC) residues in human proteins often contribute to the binding affinity of proteins, structural stability, and electrostatic interactions critical for cellular functions like molecular recognition and enzymatic activity.
Neutral Charged (PCP_NE)	Neutral charged (PCP_NE) residues in human proteins typically serve structural roles, contributing to the overall folding and stability of proteins, and may also be involved in specific protein-protein interactions or ligand binding events.

Polarity (PCP_PO)	Polarity (PCP_PO) in human proteins influences their interaction with other molecules, cellular localization, and functionality, with polar residues often found in active sites, binding interfaces, and regions crucial for protein folding and stability.
Non Polarity (PCP_NP)	Non-polarity (PCP_NP) in human proteins is typically associated with hydrophobic residues, which play essential roles in protein folding, membrane insertion, and forming stable protein-protein interactions within hydrophobic cores or lipid bilayers.
Aliphaticity (PCP_AL)	Aliphaticity (PCP_AL) in human proteins refers to the presence of aliphatic residues, such as alanine, valine, leucine, and isoleucine, which often contribute to the hydrophobic core of proteins, modulating their structural stability and interactions with other molecules.
Cyclic (PCP_CY)	Cyclic (PCP_CY) residues in human proteins, such as proline and phenylalanine, contribute to structural rigidity, protein conformational changes, and functional diversity, playing crucial roles in protein folding, stability, and interaction interfaces.
Aromaticity (PCP_AR)	Aromaticity (PCP_AR) in human proteins, characterized by residues like phenylalanine, tyrosine, and tryptophan, often mediates important protein-protein interactions, ligand binding events, and structural stability, playing key roles in various biological processes.
Acidity (PCP_AC)	Acidity (PCP_AC) in human proteins, represented by residues like aspartic acid and glutamic acid, is crucial for catalytic activity in enzymes, protein-protein interactions, and signal transduction pathways, contributing to the functional diversity and regulation of biological processes.
Basicity (PCP_BS)	Basicity (PCP_BS) in human proteins, characterized by residues such as lysine and arginine, plays essential roles in protein-protein interactions, DNA binding, and enzyme catalysis, contributing to the structural stability and functional diversity of proteins in various biological processes.
Neutral (pH) (PCP_NE_pH)	Neutral (pH) (PCP_NE_pH) residues in human proteins, such as histidine, can act as pH sensors, switching between charged and neutral states in response to changes in the cellular environment, thereby regulating protein function and signaling pathways.
Hydrophobicity (PCP_HB)	Hydrophobicity (PCP_HB) in human proteins, typically conferred by residues like alanine, isoleucine, and leucine, governs their interaction with water molecules and lipid membranes, influencing protein folding, stability, and subcellular localization in various physiological processes.
Hydrophilicity (PCP_HL)	Hydrophilicity (PCP_HL) in human proteins, often attributed to residues such as serine and glutamine, facilitates interactions with water molecules, contributing to protein solubility, structural flexibility, and the formation of protein-protein interfaces crucial for cellular functions.
Neutral (PC_NT)	Neutral (PCP_NT) residues in human proteins, like glycine, contribute to the structural flexibility and versatility of proteins, playing essential roles in protein folding, ligand binding, and molecular recognition events critical for cellular function.
Hydroxylic (PCP_HX)	Hydroxylic (PCP_HX) residues, such as serine and threonine, in human proteins play pivotal roles in post-translational modifications, protein-

	ligand interactions, and signal transduction pathways, contributing to the functional diversity and regulation of cellular processes.
Sulphur Content (PCP_SC)	Sulphur Content (PCP_SC) in human proteins, primarily represented by cysteine residues, governs the formation of disulfide bonds critical for protein stability, structural integrity, and functional regulation, playing essential roles in diverse biological processes.
Tiny (PCP_TN)	Tiny (PCP_TN) residues in human proteins, such as glycine and alanine, contribute to the compactness of protein structures, often found in protein cores or tight turns, influencing protein folding, stability, and molecular recognition events essential for cellular function.
Small (PCP_SM)	Small (PCP_SM) residues in human proteins, like alanine and serine, are commonly found in protein structures, contributing to local structural features such as loops and turns, and playing crucial roles in protein folding, stability, and molecular interactions essential for cellular function.
Large (PCP_LR)	Large (PCP_LR) residues in human proteins, such as phenylalanine and tryptophan, often occupy prominent positions in protein structures, contributing to structural stability, functional domains, and protein-protein interaction interfaces essential for cellular processes.
z1 (PCP_Z1)	The z1 (PCP_Z1) region in human proteins serves as a key regulatory site, modulating protein-protein interactions, enzymatic activities, or cellular localization, thereby influencing diverse physiological processes critical for cell function and organismal health.
z2 (PCP_Z2)	The z2 (PCP_Z2) region in human proteins plays a crucial role in structural stability, protein-protein interactions, or post-translational modifications, contributing to the intricate molecular machinery that governs cellular processes and organismal function.
z3 (PCP_Z3)	The z3 (PCP_Z3) region in human proteins is implicated in mediating dynamic conformational changes, ligand binding events, or regulatory mechanisms crucial for protein function and cellular signaling pathways, highlighting its significance in orchestrating intricate biological processes.
z4 (PCP_Z4)	The z4 (PCP_Z4) region in human proteins may serve as a structural motif, signaling domain, or post-translational modification site, contributing to the diverse functionalities and regulatory networks that underlie cellular processes and organismal homeostasis.
z5 (PCP_Z5)	The z5 (PCP_Z5) region in human proteins may act as a critical binding site for ligands, cofactors, or regulatory proteins, influencing protein function, cellular signaling pathways, and ultimately contributing to the intricate molecular landscape governing biological processes.
Secondary Structure (Helix) (PCP_SS_HE)	The Secondary Structure (Helix) (PCP_SS_HE) in human proteins, characterized by alpha helices, contributes to protein stability, structural integrity, and functional diversity, playing essential roles in biological processes such as enzyme catalysis, membrane protein insertion, and molecular recognition.
Secondary Structure (Strands) (PCP_SS_ST)	The Secondary Structure (Strands) (PCP_SS_ST) in human proteins, comprising beta strands, forms crucial elements of protein architecture, facilitating protein-protein interactions, ligand binding, and the

	formation of beta sheets critical for protein stability and molecular recognition in diverse cellular processes.
Secondary Structure (Coil) (PCP_SS_CO)	The Secondary Structure (Coil) (PCP_SS_CO) in human proteins, characterized by random coil regions, provides flexibility and versatility to protein structures, allowing for dynamic conformational changes, molecular interactions, and functional adaptations essential for diverse biological processes.
Solvent Accessibility (Buried) (PCP_SA_BU)	The Solvent Accessibility (Buried) (PCP_SA_BU) regions in human proteins are typically shielded from the surrounding solvent environment, contributing to the formation of protein cores and stabilizing interactions critical for protein folding, structural integrity, and functional specificity in cellular processes.
Solvent Accessibility (Exposed) (PCP_SA_EX)	Solvent Accessibility (Exposed) (PCP_SA_EX) regions in human proteins are readily accessible to the surrounding solvent environment, often participating in protein-protein interactions, ligand binding events, or post-translational modifications crucial for protein function and cellular signaling pathways.
Solvent Accessibility (Intermediate) (PCP_SA_IN)	Solvent Accessibility (Intermediate) (PCP_SA_IN) regions in human proteins possess a balance between buried and exposed states, contributing to structural flexibility and facilitating dynamic interactions with ligands, cofactors, or other proteins, thereby influencing diverse cellular processes and molecular functions.

```
[ ] df = pd.read_excel('mixture_3.xlsx')
```

Double-click (or enter) to edit

df.head()

	Unnamed: 0	uniprot_id	Entry Name	affects	PCP_PC	PCP_NC	PCP_NE	PCP_PO	PCP_NP	PCP_AL	...	PCP_Z2	PCP_Z3	PCP_Z4	PCP_Z5	PCP_SS_HE	P
0	0	Q95190	OAZ2_HUMAN	0	0.122	0.132	0.746	0.201	0.519	0.439	...	-0.454	-0.066	-0.409	0.209	0.392	
1	1	Q06033	ITIH3	1	0.131	0.134	0.735	0.201	0.480	0.393	...	-0.440	-0.286	-0.486	0.089	0.415	
2	2	Q9NRX6	TMEM167B	1	0.135	0.027	0.838	0.176	0.649	0.514	...	-0.774	-0.418	-0.081	-0.044	0.405	
3	3	Q9NRX3	NUA4L_HUMAN	0	0.184	0.080	0.736	0.161	0.529	0.437	...	-0.155	-0.249	-0.037	0.174	0.460	
4	4	Q02763	TIE2_HUMAN	0	0.133	0.113	0.754	0.222	0.480	0.411	...	-0.459	-0.205	-0.387	0.049	0.417	

5 rows x 34 columns

FIGURE 3: - IMPORTING DATASET IN COLAB

## BALANCING DATASET

**Imbalanced data** in machine learning occurs when one class in the dataset has significantly fewer instances than another. This can lead to biased models that perform well in the majority class but poorly in the minority class. The challenges include poor generalization to the underrepresented class and biased evaluation metrics. Resampling techniques (Under-

sampling, Over-sampling) or synthetic samples can be generated to compensate for the imbalance. The dataset we are working on is balanced at the time of creation, hence the chance of forming a biased model is low.

```
[ ] df['affects'].value_counts()

affects
1      303
0      300
Name: count, dtype: int64
```

FIGURE 4: - BALANCED DATASET

## PRE-PROCESSING OF DATA

**Data pre-processing** refers to the steps taken to clean, organize, and enhance raw data before it is fed into a machine learning model. Data needs to be organized before it can actually be effectively used for model building by applying machine learning algorithms.

Raw data is generally not available in a structured format, so all the relevant data must be collected and must be recorded in a format suitable for machine learning algorithms. There might be structural problems in the dataset or data might also be recorded incorrectly, these faulty data can affect the pattern analysis of the machine learning algorithms, so these problems need to be fixed. Domain knowledge and outlier detection techniques might come in handy in these **data cleaning** steps. Missing value or invalid value entries either need to be dropped or replaced with suitable values. Duplicates must be removed. Attributes with categorical values need to be represented with numerical values, if the categories are ordinal then the values can determine the importance, but if the data is nominal then those discrete categories can be one-hot encoded so that all the categories have equal weightage.

Often times the multiple attributes existing within the dataset have different scales and some machine learning algorithms are affected by this and give out wrong predictions, getting a faulty mapping. To avoid this situation, the data needed to be standardized or normalized.

Dimensionality reduction helps eliminate unnecessary features and improve computational time, decreases space required to store the data and also improves data visualization where there were multiple features. This helps reduce the chances of models getting over-fit. This is the curse of dimensionality, when the model tries to capture all the data-points and ends up capturing noise, resulting in a faulty model.

Dataset should also be checked for class imbalance. If there are not enough samples for the particular class, this might result in the model not properly finding ways to determine that class (biased model) and may fail to recognize that class in unseen datasets.

## MISSING AND INVALID VALUES

df.isna().sum()	
Unnamed: 0	0
uniprod_id	0
Entry Name	0
affects	0
PCP_PC	0
PCP_NC	0
PCP_NE	0
PCP_PO	0
PCP_NP	0
PCP_AL	0
PCP_CY	0
PCP_AR	0
PCP_AC	0
PCP_BS	0
PCP_NE_pH	0
PCP_HB	0
PCP_HL	0
PCP_NT	0
PCP_HX	0
PCP_SC	0
PCP_TN	0
PCP_SM	0
PCP_LR	0
PCP_Z1	0
PCP_Z2	0
PCP_Z3	0
PCP_Z4	0
PCP_Z5	0
PCP_SS_HE	0
PCP_SS_ST	0
PCP_SS_CO	0
PCP_SA_BU	0
PCP_SA_EX	0
PCP_SA_IN	0
dtype: int64	

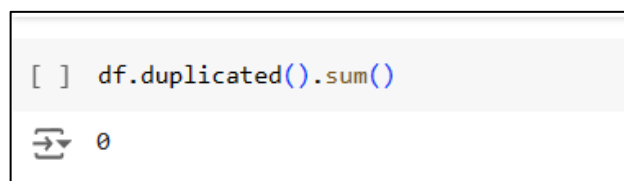
FIGURE 5: - ABSENCE OF NAN VALUES IN THE DATASET

In data processing for machine learning, it's important to deal with missing, invalid, and "NaN" (Not a Number) values in data. These issues can affect how well models learn. Fixing them, often by filling in missing info or checking for errors, helps ensure accurate predictions.



## Checking for Duplicates

Duplicate data can bias the model's learning process, leading to overfitting. Overfitting occurs when the model learns to memorize the training data rather than generalize from it. Removing duplicates helps the model focus on learning meaningful patterns in the data, leading to better generalization performance on unseen data. It also increases computational cost unnecessarily.



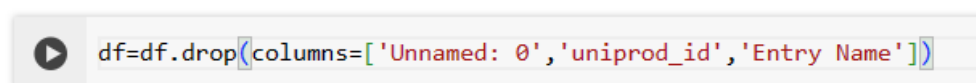
```
[ ] df.duplicated().sum()  
↔ 0
```

FIGURE 6: CHECKING FOR DUPLICATES

## Dropping Unnecessary Columns

This involves removing columns (features) from the dataset that are not relevant or necessary for the analysis or modeling task. These columns may contain redundant, irrelevant, or low-quality information that does not contribute to the predictive power of the model. Dropping unnecessary columns is a simple and straightforward data preprocessing step aimed at improving the efficiency of the model by reducing the number of features without altering the intrinsic structure of the data.

Three columns were dropped as a part of this step, namely: - Unnamed: 0, uniprot\_id, Entry Name



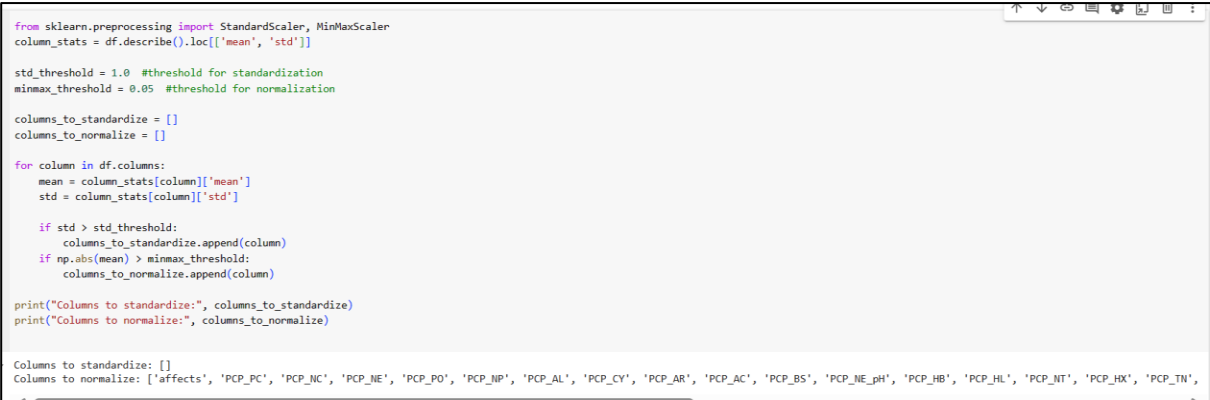
```
▶ df=df.drop(columns=['Unnamed: 0', 'uniprot_id', 'Entry Name'])
```

FIGURE 7: DROPPING UNNECESSARY COLUMNS

## Standardization and Normalization of Data

Standardization rescales the features so that they have a mean of 0 and a standard deviation of 1. This transformation ensures that the features have a similar scale and are centered around zero. Standardization is particularly useful when the features have different units or scales, and when the distribution of the data is not Gaussian (normal). It is also less affected by outliers compared to normalization.

Normalization rescales the features to a fixed range, typically between 0 and 1. This transformation preserves the relative relationships between the data points and is useful when the features have different scales but are known to be bounded within a specific range. Normalization is sensitive to outliers because it adjusts the entire range of the data based on the minimum and maximum values.



```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
column_stats = df.describe().loc[['mean', 'std']]

std_threshold = 1.0 #threshold for standardization
minmax_threshold = 0.05 #threshold for normalization

columns_to_standardize = []
columns_to_normalize = []

for column in df.columns:
    mean = column_stats[column]['mean']
    std = column_stats[column]['std']

    if std > std_threshold:
        columns_to_standardize.append(column)
    if np.abs(mean) > minmax_threshold:
        columns_to_normalize.append(column)

print("Columns to standardize:", columns_to_standardize)
print("Columns to normalize:", columns_to_normalize)
```

Columns to standardize: []  
Columns to normalize: ['affects', 'PCP\_PC', 'PCP\_NC', 'PCP\_NE', 'PCP\_PO', 'PCP\_NP', 'PCP\_AL', 'PCP\_CY', 'PCP\_AR', 'PCP\_AC', 'PCP\_BS', 'PCP\_NE\_pH', 'PCP\_HB', 'PCP\_HL', 'PCP\_NT', 'PCP\_HX', 'PCP\_TN']

**FIGURE 8: STANDARDIZATION AND NORMALIZATION OF COLUMNS**

In the above diagram, we first try to segregate the columns based on specific thresholds to see which columns need to be standardized and which ones need to be normalized. According to the thresholds set, all the columns need to be normalized but not standardized.

Although the values for all the numerical columns range from  $-1$  to  $+1$ , we include these steps as part of the standard data preprocessing steps for the sake of scalability and robustness of the models.

<pre> # Normalize and standardize the selected columns from sklearn.preprocessing import StandardScaler, MinMaxScaler for column in columns_to_normalize:     scaler = MinMaxScaler()     df[column] = scaler.fit_transform(df[[column]])  for column in columns_to_standardize:     scaler = StandardScaler()     df[column] = scaler.fit_transform(df[[column]])  df.head() </pre>																								
	affects	PCP_PC	PCP_NC	PCP_NE	PCP_PO	PCP_NP	PCP_AL	PCP_CY	PCP_AR	PCP_AC	...	PCP_Z2	PCP_Z3	PCP_Z4	PCP_Z5	PCP_SS_HE	PCP_SS_ST	PCP_SS_CO	PCP_SA_BU	PCP_SA_EX	PCP_			
0	0	0.314176	0.352000	0.583893	0.197959	0.538803	0.552273	0.472826	0.391061	0.352000	...	0.487500	0.459173	0.601606	0.558030	0.303411	0.436047	0.533333	0.517937	0.471963	0.4			
1	1	0.348659	0.357333	0.559284	0.197959	0.452328	0.447727	0.222826	0.474860	0.357333	...	0.497794	0.342524	0.539759	0.417351	0.344704	0.531977	0.416667	0.468610	0.464953	0.4			
2	1	0.363985	0.072000	0.789709	0.146939	0.827051	0.722727	0.059783	0.854749	0.072000	...	0.252206	0.272534	0.865060	0.261430	0.326750	1.000000	0.097917	0.988789	0.028037	0.2			
3	0	0.551724	0.213333	0.561521	0.116327	0.560976	0.547727	0.418478	0.525140	0.213333	...	0.707353	0.362142	0.900402	0.516999	0.425494	0.299419	0.487500	0.513453	0.404206	0.3			
4	0	0.356322	0.301333	0.601790	0.240816	0.452328	0.488636	0.293478	0.407821	0.301333	...	0.483824	0.385472	0.619277	0.370457	0.348294	0.529070	0.412500	0.513453	0.422897	0.4			

FIGURE 9: TRANSFORMATION OF COLUMNS

The segregated list of columns is passed for their respective normalization and standardization and the original columns will be transformed. Here the list for standardization is empty and all the columns are being normalized.

## DIMENSIONALITY REDUCTION

**Dimensionality Reduction** is a technique in machine learning aimed at reducing the number of input variables or features in a dataset. The goal is to simplify the dataset while retaining its essential information and patterns. High-dimensional data can suffer from the curse of dimensionality, leading to increased computational complexity, over-fitting, and difficulty in visualizing the data.

Two popular techniques for dimensionality reduction include: - Principal Component Analysis (PCA) and Linear Discriminant analysis (LDA). PCA identifies uncorrelated axes capturing maximum variance, reducing data dimensions. LDA focuses on preserving class separability, emphasizing features that discriminate between classes.

```

from sklearn.feature_selection import SelectKBest, f_classif

selector = SelectKBest(score_func=f_classif, k=20) # Select top 20 features
X_train_selected = selector.fit_transform(X_train, y_train)

selected_feature_indices = selector.get_support(indices=True)

selected_features = X_train.columns[selected_feature_indices].tolist()

importance_df = X_train[selected_features]

importance_df.shape

(482, 20)

```

FIGURE 10: FEATURE SELECTION USING SELECTKBEST

Here we are using SelectKBest for dimensionality reduction by selecting the top 20 features based on univariate statistical tests. We import the necessary libraries required for your machine learning task. We created an instance of the **SelectKBest** class. Specify the scoring function to use for ranking features. For classification tasks, we use metrics such as **f\_classif** for ANOVA F-value. Fit the **SelectKBest** instance to your data and transform the data to select the top k=20 features.

ANOVA (Analysis of Variance) F-value is a statistic used in feature selection to assess the significance of individual features in relation to the target variable. It measures the extent to which the variation in a feature's values is related to the variability in the target variable across different groups or categories.

```

plt.figure(figsize = (20, 12))

corr = importance_df.corr()
mask = np.triu(np.ones_like(corr, dtype = bool))

sns.heatmap(corr, mask = mask, linewidths = 1, annot = True, fmt = ".2f")
plt.show()

```

FIGURE 11: PLOTTING CORRELATION MATRIX

```
corr_matrix = importance_df.corr().abs()

mask = np.triu(np.ones_like(corr_matrix, dtype = bool))
tri_importance_df = corr_matrix.mask(mask)

to_drop = [x for x in tri_importance_df.columns if any(tri_importance_df[x] > 0.75)]

importance_df = importance_df.drop(to_drop, axis = 1)

print(f"The reduced dataframe has {importance_df.shape[1]} columns.")

The reduced dataframe has 12 columns.
```

*FIGURE 12: REDUCING NUMBER OF COLUMNS BASED ON CORRELATION COEFFICIENT*

We proceed to calculate the correlation matrix of the importance\_df dataframe columns. Correlation matrix helps us provide valuable insights into the strength, direction, linearity, independence, and potential multicollinearity between the features, helping to understand their relationship. Finding correlations between features can help in dimensionality reduction by identifying redundant or highly correlated features that convey similar information. Highly correlated features often represent duplicate or overlapping information in the dataset, leading to multicollinearity issues that can degrade the performance of machine learning models and increase computational complexity.

Here we set the threshold to 0.75 (in absolute value) and any pair that has more than 0.75 correlation coefficient is dropped. This gives us a resultant data frame with 12 columns.

## MODEL PREPARATION

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

## **Supervised Learning**

Supervised learning is the type of machine learning in which machines are trained using well "labelled" training data, and based on that data, machines predict the output. The labelled data means some input data is already tagged with the correct output. In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher. Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable( $x$ ) with the output variable( $y$ ).

In this project, we work on a binary classification problem dataset, where the two classes are present. Positive being denoted as 1 and negative being 0. The column named "affects" determines the class and guides the supervised machine learning algorithms.

## **Details of the Classifiers**

### **DECISION TREE**

Decision Tree is a Supervised learning technique that can be used for both classification and regression problems, but mostly it is preferred for solving classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision Tree, there are two nodes, which are the Decision Node and Leaf Node.

Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed based on features of the given dataset. It is called a Decision Tree because, like a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. To build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A Decision Tree simply asks a question and based on the answer (Yes/No), it further splits the tree into subtrees.

The Decision Tree Algorithm belongs to the family of supervised machine learning algorithms. It can be used for both a classification problem as well as for a regression problem.

The goal of this algorithm is to create a model that predicts the value of a target variable, for which the decision tree uses the tree representation to solve the problem in which the leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model.

Below are the two reasons for using the Decision Tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

In Decision Tree the major challenge is to identify the attribute for the root node in each level.

This process is known as attribute selection. We have two popular attribute selection measures:

#### 1. Information Gain:

When we use a node in a Decision Tree to partition the training instances into smaller subsets,

the entropy changes. Information gain is a measure of this change in entropy.

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy the more the information content.

## 2. Gini Index:

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower Gini index should be preferred. Sklearn supports “Gini” criteria for Gini Index and by default, it takes “gini” value.

The most notable types of Decision Tree algorithms are: -

1. IDichotomiser 3 (ID3): This algorithm uses Information Gain to decide which attribute is to be used to classify the current subset of the data. For each level of the tree, information gain is calculated for the remaining data recursively.
2. C4.5: This algorithm is the successor of the ID3 algorithm. This algorithm uses either Information gain or Gain ratio to decide upon the classifying attribute. It is a direct improvement from the ID3 algorithm as it can handle both continuous and missing attribute values.
3. Classification and Regression Tree (CART): It is a dynamic learning algorithm which can produce a regression tree as well as a classification tree depending upon the dependent variable.

## **Working:**

In a Decision Tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes



and moves further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset.
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the Decision Tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step - 3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

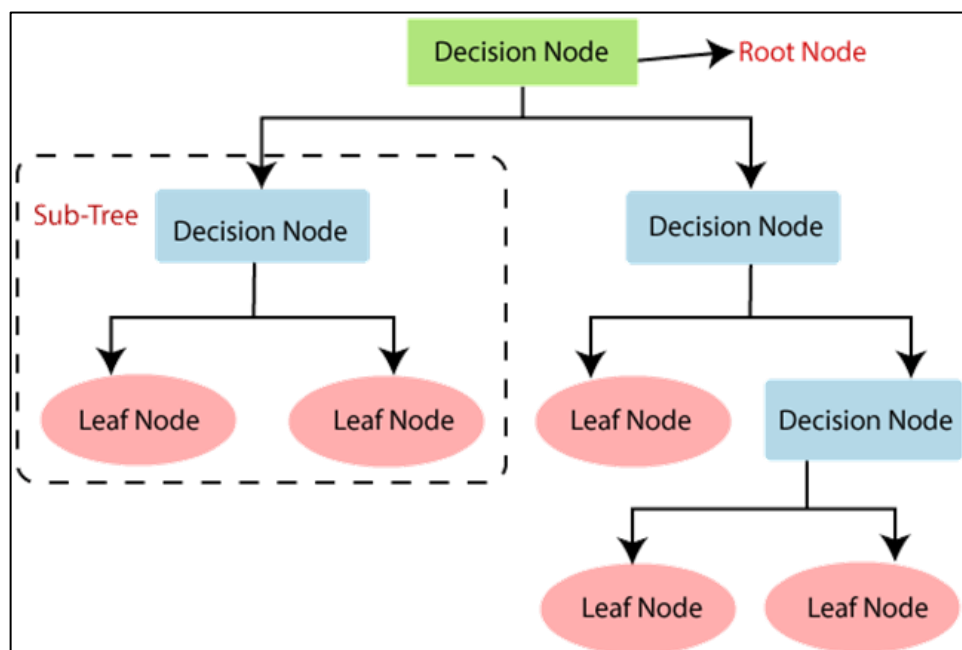


FIGURE 13: - FLOW DIAGRAM OF DECISION TREE ALGORITHM

When represented pictorially, the decision tree is a tree-like model which follows a top-down approach to assign the data to its actual target classes by the use of decision rules. It is a kind of supervised learning rule that's largely used for classification issues. Astonishingly, it works for each categorical and continuous dependent variable. During this rule, we tend to split the population into 2 or additional uniform sets. This is often done supported most important attributes/ freelance variables to create as distinct teams as attainable. Entropy: It defines the amount of uncertainty present in Dataset D.

The formula for binary classification is given by

Entropy= where  $p(i)$  is the probability of randomly selecting an example belonging to class (i).

Information Gain: On which attribute should the next splitting take place is determined by calculating Information Gain.

Gain=Entropy (D)-I (Attribute) (2)

Step to implement decision tree algorithm is given by

1. First Find the Entropy for the whole dataset, say DATASET-ENTROPY(S)
2. For each attribute/feature:
  1. Find entropy for all other attributes ENTROPY (A) to get an output.
  2. Calculate the consider attribute Average information Entropy.
3. Then find gain for the attribute which is considered.
3. Select the best gain attribute.
4. Repeat the above steps to get a suitable output.

## **RANDOM FOREST**

Random Forest is a supervised learning algorithm. It is an extension of machine learning classifiers which include the bagging to improve the performance of Decision Tree. It is an ensemble learning algorithm. It combines three predictors, and trees are dependent on a random vector which is independently sampled. The distribution of all trees are the same. Random Forests splits nodes using the best among of a predictor subset that are randomly chosen from the node itself, instead of splitting nodes based on the variables. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest consists of trees. It is said that the more trees it has, the more robust a forest is. Random Forests create Decision Trees on randomly selected data samples, get predictions from each tree and select the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random Forests have a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction

from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

### **Assumptions**

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random Forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

### **Algorithm Steps:**

It works in four steps:

- Select random samples from a given dataset.
- Construct a Decision Tree for each sample and get a prediction result from each Decision Tree.
- Perform a vote for each predicted result.
- Select the prediction result with the most votes as the final prediction.

### **Advantages:**

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

## Disadvantages:

Although Random Forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

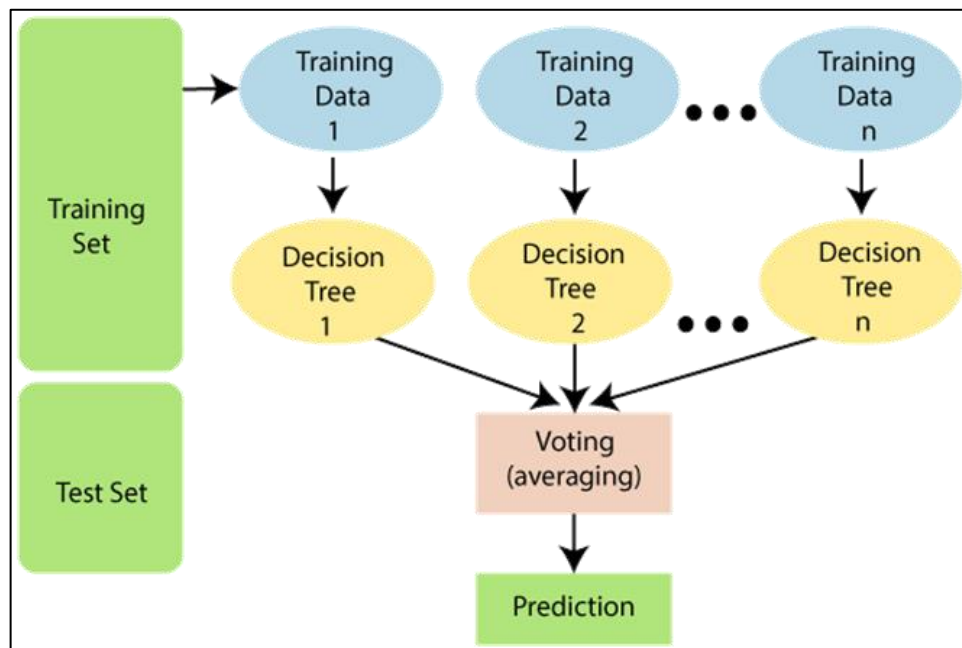


FIGURE 14: - FLOW DIAGRAM FOR RANDOM FOREST ALGORITHM

## GRADIENT BOOSTING

Gradient Boosting is a machine learning technique used for both regression and classification tasks. It works by combining multiple weak learners, typically decision trees, into a strong learner.

Gradient Boosting builds models sequentially, with each new model correcting errors made by the previous ones. It minimizes a loss function, such as the residual error in regression or the negative log-likelihood in classification, by adding new models that focus on the examples the previous models got wrong.

If we look at the gradient descent method, we find that it is an iterative optimization algorithm used to minimize a given loss function or cost function. In machine learning, we use it to update the parameters of a model or find the optimal values that minimize the difference between predicted and actual values.

Gradient descent method is an iterative optimization algorithm used to minimize a given loss function or cost function. In machine learning, we use it to update the parameters of a model or find the optimal values that minimize the difference between predicted and actual values.

The following step this algorithm will follow:

1. It will start with training a base model (usually a decision tree). This model gives predictions for the entire dataset.
2. After making predictions, using the original values, it calculates the errors. These errors can be thought of as the parts of  $y$  that are still unexplained by the base model.
3. Before applying the next sequenced model, the algorithm applies the gradient descent method to calculate the optimized parameters of the next model.
4. Train a new model, and This model is fitted to minimize the difference between the predicted residuals and the actual residuals.
5. After training a new model, this algorithm calculates the new residuals by subtracting the updated predictions from the actual values of  $y$ . These residuals represent the remaining unexplained variability in the target variable.
6. This whole process gets repeated iteratively, adding the new models to the ensemble while applying gradient descent on it and updating the predictions and residuals.
7. At the final stage, the method combines the predictions from all the models where we consider the final predictions refined estimation of  $y$ .

**Strengths:**

1. High Accuracy: Gradient Boosting often yields highly accurate predictions, making it suitable for a wide range of tasks.
2. Handles Mixed Data Types: It can handle a mixture of categorical and numerical features without extensive preprocessing.
3. Feature Importance: It provides insights into feature importance, helping in feature selection and understanding the data.
4. Robustness to Overfitting: It includes regularization techniques like shrinkage and pruning, which help prevent overfitting.

**Weaknesses:**

1. Computationally Intensive: Training Gradient Boosting models can be computationally expensive, especially with large datasets or complex models.
2. Sensitive to Noise and Outliers: It can be sensitive to noisy data or outliers, which may lead to overfitting.
3. Requires Tuning: Proper tuning of hyperparameters, such as learning rate and tree depth, is essential for optimal performance.
4. Less Interpretable: Compared to simpler models like linear regression, Gradient Boosting models are less interpretable, making it challenging to understand the underlying relationships in the data.

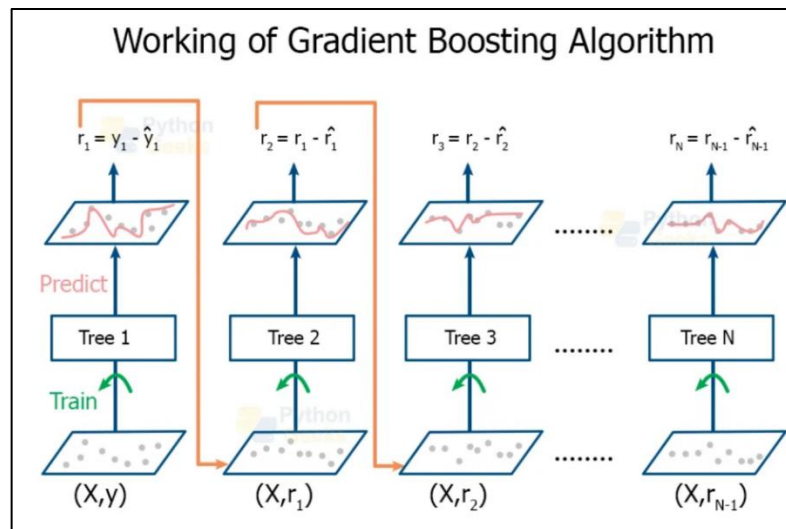


FIGURE: 15: FLOW DIAGRAM OF GRADIENT BOOSTING

## SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.



The followings are important concepts in SVM -

Support Vectors - Data Points that are closest to the hyperplane are called support vectors.

Separating line will be defined with the help of these data points.

Hyperplane - As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.

Margin - It may be defined as the gap between two lines on the closest data points of different classes. It can be calculated as the perpendicular distance from the line to the 12 support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

### **Types of SVM:**

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier. The objective of the support vector machine algorithm is to find a hyperplane in an  $N$  dimensional space ( $N$  - the number of features) that distinctly classifies the data points.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where the number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also

memory efficient.

- Versatile: different kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

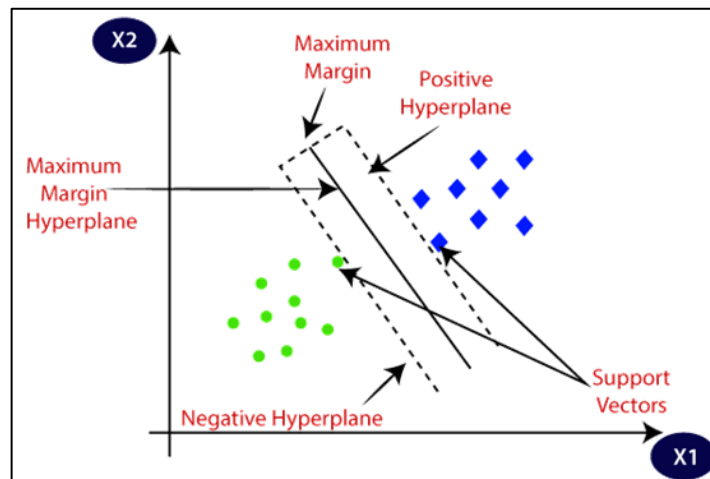


FIGURE 16: - WORKING MODEL OF SVM ALGORITHM

## LOGISTIC REGRESSION

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

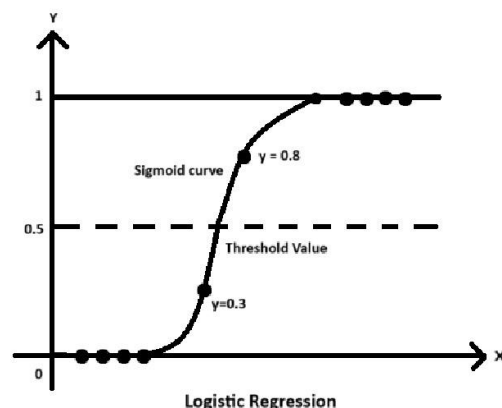
Logistic regression predicts the output of a categorical dependent variable. Therefore, the

outcome must be of a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Logistic Regression is much similar to Linear Regression except for how they are used. Linear Regression is used for solving Regression problems, whereas logistic regression is used for solving the classification problems. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

### **Advantages:**

Logistic Regression is one of the simplest machine learning algorithms and is easy to implement yet provides great training efficiency in some cases. Also due to these reasons, training a model with this algorithm doesn't require high computation power.

The predicted parameters (trained weights) give inference about the importance of each feature. The direction of association i.e. positive or negative is also given. So we can use Logistic Regression to find out the relationship between the features.



*FIGURE 17: - SIGMOID CURVE FOR LOGISTIC REGRESSION*

## The Logistic Curve

The logistic curve relates the independent variable,  $X$ , with  $Y$  (independent variable) by determining probability. The formula to do so may be written either

$$P = \frac{1}{1 + e^{-(a+bX)}}$$

where  $P$  is the probability of a 1 (the proportion of 1s, the mean of  $Y$ ),  $e$  is the base of the natural logarithm (about 2.718), and  $a$  and  $b$  are the parameters of the model.

## XGBOOST

XG-boost is an implementation of Gradient Boosted decision trees. It is a type of Software library that was designed basically to improve speed and model performance. In this algorithm, decision trees are created in sequential form. Weight plays an important role in XG-boost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. Weight of variables predicted wrong by the tree is increased and these the variables are then fed to the second decision tree. These individual classifiers/predictors then assemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined predict.

1. Regularization: XG-boost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization which prevents the model from overfitting. That is why, XG-boost is also called regularized form of GBM (Gradient Boosting Machine). While using Scikit Learn library, we pass two hyper-parameters (alpha and lambda) to XG-boost related to regularization. alpha is used for L1 regularization and lambda is used for L2 regularization.

2. **Parallel Processing:** XG-boost utilizes the power of parallel processing and that is why it is much faster than GBM. It uses multiple CPU cores to execute the model. While using Scikit Learn library, nthread hyper-parameter is used for parallel processing. nthread represents number of CPU cores to be used. If you want to use all the available cores, don't mention any value for nthread and the algorithm will detect automatically.
3. **Handling Missing Values:** XG-boost has an in-built capability to handle missing values. When XG-boost encounters a missing value at a node, it tries both the left and right hand split and learns the way leading to higher loss for each node. It then does the same when working on the testing data.
4. **Cross Validation:** XG-boost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run. This is unlike GBM where we must run a grid-search and only a limited values can be tested.
5. **Effective Tree Pruning:** A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm. XG-boost on the other hand make splits upto the max\_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.

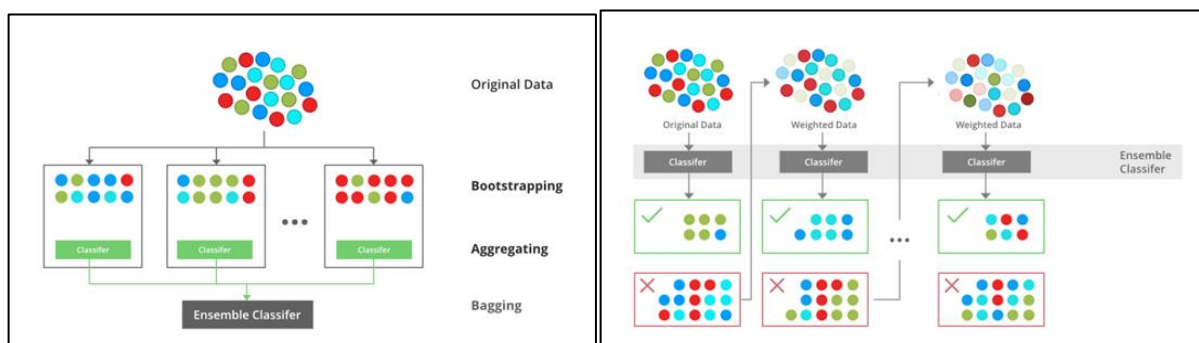


FIGURE 18: - WORKING DIAGRAM FOR XGBOOST ALGORITHM

XGboost is an implementation of the ensemble learning algorithm boosting. The fundamental principle of the Xgboost is to train the model using residuals. The outcome of the most recent tree training is utilized as the input for the subsequent iteration, and the error is progressively decreased over numerous serial iterations. Finally, all weak learners are linearly weighted to produce the ensemble learner.

Additionally, when training the Xgboost tree, the effective splitting point is chosen using an information-gain-based greedy algorithm.

## **NAÏVE BAYES**

Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset.

Naive Bayes Classifier is one of the simplest and most effective Classification algorithms which helps in building fast machine learning models that can make quick predictions.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

The Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

The Naive Bayes algorithm is comprised of two words Naive and Bayes, Which can be described as:

- Naive: It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the basis of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem. Bayes's theorem: Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

The formula for Bayes' theorem is given as: 
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,  $P(A|B)$  is Posterior probability: Probability of hypothesis A on the observed event B.  
 $P(B|A)$  is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

$P(A)$  is Prior Probability: Probability of hypothesis before observing the evidence.

$P(B)$  is Marginal Probability: Probability of Evidence.

The main base of this classifier is Bayes Theorem which is computed using conditional probability. This classifier computes the probability of occurrence of the output feature given a set of input attribute values. Internally, the probability of every value with respect to the feature will be calculated. After calculating the probability of all the values of the output attribute, we consider the value which has maximum probability. The naïve Bayes model is extremely simple and fast for large datasets and works well for binary classification.

## GAUSSIAN NAÏVE BAYES

Gaussian Naive Bayes is a classification algorithm based on Bayes' theorem, assuming that features follow a normal (Gaussian) distribution and are independent of each other. Gaussian Naive Bayes is a specific type of Naive Bayes algorithm that assumes that features follow a Gaussian distribution (also known as normal distribution). It is suitable for continuous features where the values of each feature are assumed to be sampled from a Gaussian distribution. It calculates the mean and standard deviation of each feature for each class and uses these parameters to estimate the likelihood of observing a particular feature value given a class. Often used in classification tasks where the features are continuous variables.

### **Strengths:**

1. Simple and efficient: Easy to implement and works well with high-dimensional data.
2. Handles continuous data: Suitable for datasets with continuous features that can be modeled using a Gaussian distribution.
3. Performs well with small datasets: Still effective even with limited training data.

### **Weaknesses:**

1. Assumes feature independence: Might not hold true in real-world datasets.
2. Sensitive to outliers: Can be influenced by extreme values in the data.
3. Limited expressiveness: May not capture complex relationships between features.



## K NEAREST NEIGHBOURS (KNN)

K-Nearest Neighbors (KNN) is a machine learning algorithm used for classification and regression tasks. It operates by considering the proximity of data points in a feature space. In KNN, a data point is classified or predicted by the majority class or average of its k-nearest neighboring data points, where 'k' is a user-defined parameter. In essence, KNN assumes that data points that are close to each other in the feature space share similar characteristics. KNN is a simple and intuitive method, making it valuable for various applications, especially when the relationship between variables is not well-defined or when patterns in the data are not linear. It's a versatile tool for tasks like data classification, regression, and recommendation systems.

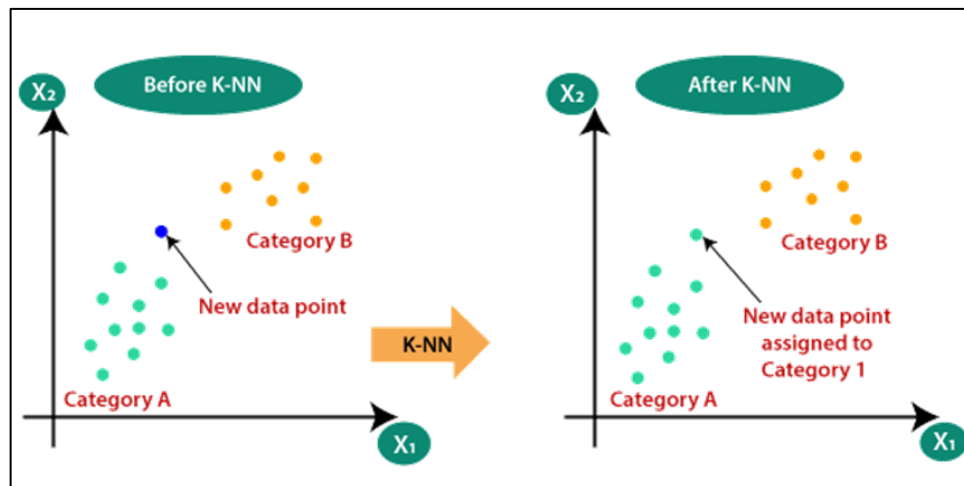


FIGURE 19: - WORKING OF KNN ALGORITHM

### Algorithmic steps for k-means clustering:

Let  $X = \{x_1, x_2, x_3, \dots, x_n\}$  be the set of data points and  $V = \{v_1, v_2, \dots, v_c\}$  be the set of centers.

- 1) Randomly select 'c' cluster centers.
- 2) Calculate the distance between each data point and cluster centers. Assign the data point to

the cluster center whose distance from the cluster center is minimum of all the cluster centers..

3) Recalculate the new cluster center using: 
$$v_i = \left(\frac{1}{c_i}\right) \sum_{j=1}^{c_i} x_i$$

where, 'ci' represents the number of datapoints in 50th cluster.

4) Recalculate the distance between each data point and newly obtained cluster centers.

5) If no data point was reassigned then stop, otherwise repeat from step 3.

## ADABOOST

Adaboost, short for Adaptive Boosting, is a machine learning algorithm that combines multiple weak classifiers to create a strong classifier. It works by iteratively training weak models on subsets of the data, where each subsequent model focuses more on the instances that the previous ones misclassified. The final model is a weighted sum of these weak models, where each model's weight depends on its accuracy. In a nutshell, we can say that adaptive boosting is a way to reduce the error of any machine-learning algorithm, which work by aligning many weak machine-learning models into one strong machine-learning model.

The steps involved in its working are as follows: -

- 1) Adaboost starts with a simple model, often a decision tree that just asks one question about the data. This simple model is called a "weak learner."
- 2) Adaboost looks at where this simple model makes mistakes. It pays more attention to the data points it got wrong.
- 3) Adaboost then builds another simple model, but this time it pays extra attention to the data points that the first model got wrong. This new model also focuses on getting those points right.

- 4) Adaboost combines these models together, giving more weight to the ones that performed well overall and less weight to the ones that didn't.
- 5) Adaboost repeats this process, each time building a new model that focuses on the mistakes of the combined models before it.
- 6) When it's time to make a prediction on new data, Adaboost combines the predictions from all the models it built. It gives more weight to the predictions of the models that performed better during training.

### **Strengths:**

1. Adaboost is highly accurate and often outperforms other algorithms.
2. It's versatile and can be used with various base classifiers.
3. It handles complex data well and is less prone to overfitting.

### **Weaknesses:**

1. Adaboost can be sensitive to noisy data and outliers, which may affect its performance.
2. It's computationally expensive, especially when dealing with large datasets or complex models.
3. It may suffer from the problem of overfitting if the base classifiers are too complex or the number of iterations is too high.

## **EXTRA TREES**

Extra Trees, short for Extremely Randomized Trees, is a machine learning algorithm that belongs to the ensemble learning family. Similar to Random Forests, Extra Trees builds multiple decision trees and combines their predictions to make more accurate and robust

predictions. However, unlike Random Forests, Extra Trees randomly selects feature subsets for each decision tree and further randomizes the threshold for feature splitting, making the trees “extremely randomized.” This extra randomness helps reduce variance and overfitting, making Extra Trees less sensitive to noise and outliers in the data. In Random Forest, each decision tree is trained on a bootstrap sample of the training data (sampling with replacement), and at each node of the tree, a random subset of features is considered for splitting. Additionally, because of its randomization, Extra Trees can be computationally faster to train compared to Random Forests.

### **Strengths:**

1. **Reduced Variance:** Extra Trees further reduces the variance compared to Random Forest by adding an additional layer of randomness in the tree-building process.
2. **Less Prone to Overfitting:** The extra randomness in feature selection and node splitting helps prevent overfitting, making it robust to noisy data.
3. **Computationally Efficient:** Extra Trees are faster to train than traditional decision trees or Random Forest because they randomly select features at each split, reducing the computational burden.
4. **Parallelization:** The construction of individual trees in Extra Trees is independent, allowing for easy parallelization, making it suitable for large datasets.

### **Weaknesses:**

1. **Bias:** The extra randomness in Extra Trees can lead to increased bias compared to Random Forest, which may affect predictive performance, especially in datasets with complex relationships.
2. **Lack of Interpretability:** Like other ensemble methods, interpreting the results of Extra Trees

can be challenging due to the ensemble nature of the algorithm.

3. Difficulty in Tuning: While Extra Trees are less sensitive to hyperparameters compared to Random Forest, selecting the optimal number of trees and other parameters can still be challenging.

4. Limited Feature Importance Analysis: The randomness in feature selection can make it harder to assess the importance of individual features in the model.

## **LINEAR SUPPORT VECTOR MACHINE**

Linear SVM aims to find the optimal hyperplane that maximizes the margin between classes. It does this by identifying support vectors, which are the data points closest to the hyperplane, and uses them to define the decision boundary. Linear Support Vector Machine is a specific type of SVM where the decision boundary is linear. LSVM focuses on finding the linear hyperplane that best separates the classes while maximizing the margin between them. LSVM also maximizes the margin between the classes, but it does so specifically for linear decision boundaries. It seeks to find the linear hyperplane that separates the classes with the maximum margin.

### **Strengths:**

1. Effective in High-Dimensional Spaces: Linear SVM performs well even in high-dimensional spaces, making it suitable for tasks with many features.

2. Memory Efficient: It uses only a subset of training points (support vectors) in the decision function, making it memory efficient, especially with large datasets.

3. Effective for Linearly Separable Data: When the data is linearly separable, Linear SVM can

find a clear decision boundary with a wide margin, leading to good generalization performance.

4. Robust to Overfitting: It includes regularization parameters that help prevent overfitting, making it robust against noisy data.

### **Weaknesses:**

1. Not Suitable for Non-Linear Data: Linear SVM performs poorly on non-linear data since it can only find linear decision boundaries.

2. Sensitive to Outliers: Outliers can significantly affect the position of the hyperplane, potentially leading to suboptimal performance.

3. Requires Feature Scaling: Linear SVM works best when features are scaled, so preprocessing such as normalization or standardization may be necessary.

4. Limited Interpretability: While Linear SVM provides a clear decision boundary, it may be less interpretable compared to other models like decision trees.

## **STOCHASTIC GRADIENT DESCENT**

Stochastic Gradient Descent (SGD) is a popular optimization algorithm used in machine learning for training models, especially in large-scale datasets. It's a variant of the gradient descent algorithm, where instead of computing the gradient using the entire dataset, it uses only a small subset, making it faster and more scalable. Stochastic Gradient Descent (SGD) is an optimization algorithm commonly used for training machine learning models, especially in large-scale and online learning settings. Unlike traditional gradient descent, which computes the gradient of the loss function using the entire training dataset, SGD updates the model parameters iteratively using a single training example or a small subset (mini-batch) of training examples at each iteration. This stochastic nature allows SGD to make frequent updates to the

parameters, making it computationally efficient and suitable for large datasets. By following the gradient direction, SGD gradually descends toward the minimum of the loss function, aiming to find the optimal set of parameters that minimize the prediction error.

### **Strengths:**

1. Efficiency: SGD is computationally efficient because it processes small batches of data at a time, making it suitable for large datasets and online learning scenarios.
2. Scalability: It can handle large-scale datasets efficiently, making it useful in scenarios where traditional gradient descent methods may struggle.
3. Regularization: SGD naturally incorporates regularization techniques like L1 and L2 regularization, helping prevent overfitting.
4. Versatility: SGD can be used with various loss functions, making it adaptable to different types of machine learning problems.

### **Weaknesses:**

1. Noisy Updates: Because SGD updates the model parameters based on small subsets of data, it can introduce more noise compared to batch gradient descent, leading to slower convergence.
2. Sensitivity to Learning Rate: The performance of SGD is sensitive to the learning rate hyperparameter, which needs to be carefully tuned for optimal results.
3. Local Minima: Like gradient descent, SGD is prone to getting stuck in local minima, especially in non-convex optimization problems.
4. Complexity: Tuning SGD can be complex, as it involves selecting appropriate learning rates, batch sizes, and regularization parameters, which may require experimentation.

## **BAGGING CLASSIFIER**

The Bagging Classifier, short for Bootstrap Aggregating, is an ensemble learning technique designed to improve the stability and accuracy of machine learning models, particularly decision trees. It operates by training multiple base learners (typically decision trees) on different subsets of the training data, sampled with replacement. Each base learner is trained independently, resulting in a diverse set of models. During prediction, the Bagging Classifier aggregates the predictions from all base learners using averaging (for regression) or voting (for classification).

### **Strengths:**

1. Reduces Variance: Bagging helps in reducing overfitting by averaging the predictions of multiple models trained on different subsets of data, which leads to more stable and reliable predictions.
2. Handles High-Dimensional Data: It can effectively handle high-dimensional datasets with a large number of features without requiring feature selection or dimensionality reduction techniques.
3. Robustness: Bagging is robust to outliers and noise in the data due to its averaging effect, making it suitable for noisy datasets.
4. Versatility: It can be applied to a variety of machine learning algorithms as base learners, providing flexibility in model selection.

### **Weaknesses:**

1. Increased Complexity: The ensemble of multiple models increases the computational complexity and memory requirements compared to training a single model, especially for large



datasets.

2. Limited Interpretability: The combination of multiple models makes it challenging to interpret the overall decision-making process, as the contribution of each base learner is not easily discernible.
3. No Improvement for Biased Base Learners: If the base learner used in bagging is inherently biased, bagging may not significantly improve the model's performance, as it relies on the diversity of base learners for its effectiveness.
4. Training Time: Training multiple models can be time-consuming, especially for complex models or large datasets, although this can be mitigated through parallelization techniques.

```
classifiers = [  
    ("Random Forest", RandomForestClassifier(), {"classifier__n_estimators": [100, 50,150,200,250, 300]}),  
    ("Gradient Boosting", GradientBoostingClassifier(), {"classifier__n_estimators": [50,100,150, 200,250, 300,350]}),  
    ("Support Vector Machine", SVC(), {"classifier__C": [0.1, 1.0, 10.0], "classifier__kernel": ["linear", "rbf", "poly"]}),  
    ("Logistic Regression", LogisticRegression(), {"classifier__C": [0.1, 1.0, 10.0], "classifier__solver": ["liblinear", "saga"]}),  
    ("XGBoost", xgb.XGBClassifier(), {"classifier__n_estimators": [100, 150, 200, 300], "classifier__learning_rate": [0.1, 0.001, 0.01, 0.001]}),  
    ("Naive Bayes", GaussianNB(), {}),  
    ("K-Nearest Neighbors (KNN)", KNeighborsClassifier(), {"classifier__n_neighbors": [3,4, 5, 7,8], "classifier__weights": ["uniform", "distance"]}),  
    ("Decision Tree", DecisionTreeClassifier(), {"classifier__max_depth": [None, 5, 10]}),  
    ("AdaBoost", AdaBoostClassifier(), {"classifier__n_estimators": [50, 100, 150], "classifier__learning_rate": [0.1,0.3,1.0]}),  
    ("Gaussian Naive Bayes", GaussianNB(), {}),  
    ("Extra Trees", ExtraTreesClassifier(), {"classifier__n_estimators": [100, 200, 300]}),  
    ("Linear Support Vector Machines", LinearSVC(), {"classifier__C": [0.1, 1.0, 10.0]}),  
    ("Stochastic Gradient Descent (SGD)", SGDClassifier(), {"classifier__loss": ["hinge", "log", "modified_huber"]}),  
    ("Bagging Classifier", BaggingClassifier(), {"classifier__n_estimators": [10, 20, 30]}),  
]
```

FIGURE 20: CLASSIFIERS USED IN THIS PROJECT

## Cross Validation

Cross-validation is like giving a model multiple quizzes to ensure it truly understands the data. Instead of relying on a single training-test split, cross-validation involves dividing the data into multiple sets, training the model on different combinations, and averaging the performance. This ensures a more robust evaluation and reliable model assessment.

## Hyperparameter Tuning

Hyperparameter tuning in machine learning is the process of finding the optimal configuration for the hyperparameters of a model. Hyperparameters are settings that are not learned from the data but are set before the training process. Examples include learning rates, tree depths, and regularization strengths. Tuning is crucial because the right hyperparameter values can significantly impact a model's performance. It involves systematic experimentation, often using techniques like Grid Search or Random Search, to discover the combination of hyperparameter values that leads to the best model accuracy and generalization on new, unseen data.

## GridSearch CV

GridSearchCV is a helpful tool in machine learning that automates the process of finding the best combination of model hyperparameters. It systematically tests various parameter values from a predefined grid, allowing the selection of the most effective settings for optimal model performance, saving time and improving accuracy.

## Pipeline

In Python, particularly in the context of machine learning (ML), a **pipeline** is a tool provided by libraries like **scikit-learn** to streamline and simplify the process of building and evaluating models. It allows you to chain together a sequence of data processing steps and a model into a single object, which can be treated as one cohesive unit. This is especially useful for ensuring that all steps are applied consistently and correctly, both during training and testing phases. A pipeline in **scikit-learn** is essentially a linear sequence of steps where each step is either a transformer (e.g., scaler, encoder) or an estimator (e.g., classifier, regressor). These steps are applied sequentially, ensuring that the same transformations are applied to both the training and testing data. It simplifies hyper-parameter tuning by allowing you to define a parameter grid that includes parameters from both the preprocessing steps and the model itself. Pipelines integrate seamlessly with these tools, allowing you to search over parameters of both the transformers and the final estimator in a single step.

```

# Perform hyperparameter tuning for each classifier
for classifier, clf, param_grid in classifiers:
    pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("classifier", clf)
    ])
    grid_search = GridSearchCV(pipeline, param_grid, cv=5)
    grid_search.fit(X_train, y_train)

    best_estimator = grid_search.best_estimator_

```

FIGURE 21: USING PIPELINE AND GRIDSEARCHCV FOR HYPER-PARAMETER TUNING

## Train-Test split

The dataset is split into test and train datasets, in the ratio 8:2 where 8 parts out of 10 is the train data and 2 parts is the test data. The train data is used to train the machine learning model I.e. the model figures out the relationship or underlying pattern and maps the relation between the independent and dependent variable. After the model has been trained, the test data is used as the unknown unseen data and is used to check the validity of the trained model. We use various metrics to determine that, which will be discussed later.

```

[ ] from sklearn.model_selection import train_test_split
X = df.drop('affects', axis=1) # Input variables (features)
y = df['affects'] # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=False)

```

FIGURE 22: - CODE FOR DIVIDING THE DATASET FOR TRAIN-TEST SPLIT

## Prediction And Testing

After the models are trained using various algorithms, they are tested with an unknown dataset to see if the model can correctly classify the data. Various performance metrics are used to determine the robustness of the model. We compare the predictions made by the model with the actual outputs and see how many were classified correctly and how many went wrong, according to which the model is optimized and tuned until the metrics improve.

```
grid_search.fit(X_train, y_train)

best_estimator = grid_search.best_estimator_

# Evaluate the best estimator on the test set
y_pred = best_estimator.predict(X_test)
```

*FIGURE 23: - CODE FOR PREDICTION BY MODELS*

## CHAPTER 5

# RESULTS AND DISCUSSION

### Analysis of Dataset and classifiers

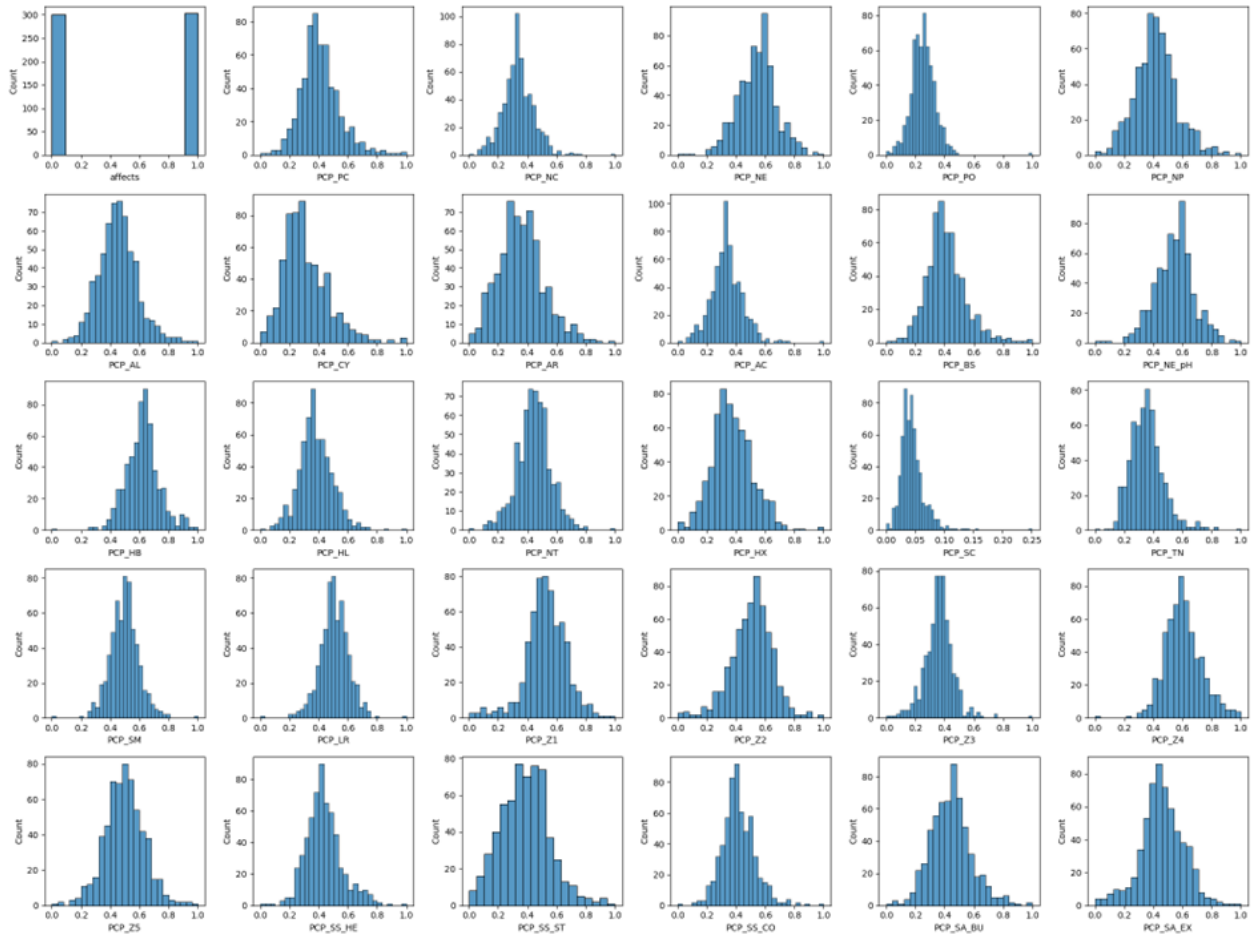


FIGURE 24: - HISTOGRAM FOR THE VALUES OF DIFFERENT ATTRIBUTES

Table 2: - Summary Statistics for DataFrame Columns

Attribute Name	Count	Mean	Standard	Minimum	25%	50%	75%	Max
affects	603	0.502	0.500	0.000	0.000	1.000	1.000	1.000
PCP_PC	603	0.146	0.035	0.040	0.124	0.142	0.164	0.301
PCP_NC	603	0.125	0.040	0.000	0.102	0.122	0.148	0.375
PCP_NE	603	0.728	0.59	0.485	0.689	0.733	0.763	0.932
PCP_PO	603	0.227	0.041	0.104	0.203	0.228	0.251	0.594
PCP_NP	603	0.464	0.065	0.276	0.422	0.461	0.500	0.727

PCP_AL	603	0.395	0.059	0.196	0.357	0.394	0.428	0.636
PCP_CY	603	0.059	0.028	0.003	0.040	0.054	0.075	0.187
PCP_AR	603	0.073	0.028	0.009	0.055	0.072	0.089	0.188
PCP_AC	603	0.125	0.040	0.000	0.102	0.122	0.148	0.375
PCP_BS	603	0.146	0.035	0.040	0.124	0.142	0.164	0.301
PCP_NE_pH	603	0.728	0.059	0.485	0.689	0.733	0.763	0.932
PCP_HB	603	0.473	0.062	0.142	0.436	0.475	0.507	0.676
PCP_HL	603	0.241	0.040	0.112	0.217	0.238	0.263	0.454
PCP_NT	603	0.369	0.051	0.176	0.340	0.371	0.399	0.613
PCP_HX	603	0.129	0.030	0.048	0.109	0.127	0.149	0.267
PCP_SC	603	0.044	0.021	0.000	0.032	0.041	0.054	0.251
PCP_TN	603	0.238	0.051	0.080	0.203	0.235	0.264	0.526
PCP_SM	603	0.491	0.062	0.142	0.436	0.475	0.507	0.676
PCP_LR	603	0.508	0.062	0.187	0.473	0.507	0.547	0.824
PCP_Z1	603	0.270	0.366	1.080	0.085	0.263	0.505	1.493
PCP_Z2	603	-0.430	0.194	-1.117	-0.548	-0.416	-0.311	0.243
PCP_Z3	603	-0.249	0.184	-0.932	-0.345	-0.241	-0.153	0.945
PCP_Z4	603	-0.405	0.157	-1.158	-0.509	-0.420	-0.316	0.087
PCP_Z5	603	0.156	0.116	-0.267	0.083	0.155	0.229	0.586
PCP_SS_HE	603	0.464	0.069	0.223	0.421	0.457	0.500	0.780
PCP_SS_ST	603	0.247	0.057	0.115	0.207	0.248	0.285	0.459
PCP_SS_CO	603	0.287	0.054	0.088	0.254	0.283	0.321	0.568
PCP_SA_BU	603	0.405	0.062	0.208	0.364	0.405	0.439	0.654
PCP_SA_EX	603	0.333	0.063	0.137	0.299	0.333	0.375	0.565
PCP_SA_IN	603	0.262	0.045	0.119	0.243	0.260	0.291	0.443

As we can see that the columns don't vary too much by scale so there's no need to standardize but normalization might be helpful in cases of some models, so we will subject all the columns to a threshold of 1.0 for standardization and 0.05 for normalization. If the standard coefficient of the column is greater than the threshold then we note it down for standardization and if it has a mean greater than the normalization threshold, we add it for normalization.

After subjecting all the columns to the condition, it is seen that none of the columns needs standardization and all of them require normalization.

We use correlational matrix for determining the interdependence between the columns and the columns which have a correlational coefficient greater than 0.75 are dropped as they are too dependent on each other, most probably in a linear fashion. Hence, it won't provide much information. We settle with 12 columns for training and testing this way.

## Model Formation

### 1) Random Forest Classifier

Hyperparameter considered: - `classifier_n_estimators` (The `parameter` in a Random Forest algorithm specifies the number of decision trees that the model should build and ensemble to make its final predictions.)

Values considered for hyper-parameters: - 10,20,30,40,100,70,50,150,200,250, 300

The best value for the hyper-parameter selected by GridSearchCV: - 300

### 2) Gradient Boost Classifier

Hyperparameter considered: - `classifier_n_estimators` (The ``classifier__n_estimators`` parameter in a Gradient Boosting algorithm determines the number of boosting stages (or iterations) the model will perform, effectively controlling the number of individual trees that will be sequentially added to the ensemble.)

Values considered for hyper-parameters: - 10,20,30,40,100,70,50,150,200,250, 300, 350

The best value for the hyper-parameter selected by GridSearchCV: - 50

### 3) Support Vector Machine Classifier

Hyperparameter considered: - `classifier_C`, `classifier_kernel` (In a Support Vector Machine (SVM) algorithm, the ``classifier__C`` parameter controls the trade-off between achieving a low error on the training data and minimizing the model complexity, while the ``classifier__kernel`` parameter specifies the kernel type to be used in transforming the data to find an optimal boundary.)

Values considered for hyper-parameters (`classifier_C`): - 0.001, 0.01, 100, 1000, 0.1, 1.0, 10.0

Values considered for hyper-parameters (`classifier_kernel`): - "linear", "rbf", "poly"

The best value for the hyper-parameter selected by GridSearchCV(`classifier_C`): - 0.1

The best value for the hyper-parameter selected by GridSearchCV: - rbf

### 4) Logistic Regression Classifier

Hyperparameter considered: - `classifier_C`, `classifier_solver` (In a Logistic Regression algorithm, the ``classifier__C`` parameter controls the regularization strength, with smaller

values indicating stronger regularization, while the `classifier\_\_solver` parameter specifies the algorithm used for optimization of the logistic regression objective function.)

Values considered for hyper-parameters (classifier\_C): - 0.001, 0.01, 100, 1000, 0.1, 1.0, 10.0

Values considered for hyper-parameters (classifier\_solver): - "liblinear", "saga"

The best value for the hyper-parameter selected by GridSearchCV(classifier\_C): - 0.1

The best value for the hyper-parameter selected by GridSearchCV(solver): - liblinear

## 5) XGBoost Classifier

Hyperparameter considered: - classifier\_n\_estimators, classifier\_learning\_rate (In an XGBoost algorithm, the `classifier\_\_n\_estimators` parameter specifies the number of boosting rounds (or trees) to be added to the model, and the `classifier\_\_learning\_rate` parameter controls the contribution of each tree by scaling its weights, balancing the trade-off between the number of estimators and the impact of each individual estimator.)

Values considered for hyper-parameters (n\_estimators): - 10,20,30,40,100,70,50,150,200,250, 300

Values considered for hyper-parameters(learning\_rate): - 0.1, 0.001, 0.01, 0.0001

The best value for the hyper-parameter selected by GridSearchCV(n\_estimator): - 200

The best value for the hyper-parameter selected by GridSearchCV(learning\_rate): - 0.01

## 6) K Nearest Neighbours Classifier

Hyperparameter considered: - classifier\_n\_neighbours, classifier\_weights(The parameter in a Random Forest algorithm specifies the number of decision trees that the model should build and ensemble to make its final predictions.)

Values considered for hyper-parameters (n\_neighbours): - 2,3,4, 5,6, 7,8,9,10,11

Values considered for hyper-parameters (weights): - "uniform", "distance"

The best value for the hyper-parameter selected by GridSearchCV (n\_neighbours): - 5

The best value for the hyper-parameter selected by GridSearchCV(weights): - distance



## 7) Decision Tree Classifier

Hyperparameter considered: - classifier\_max\_depth (The `classifier\_\_max\_depth` parameter in a Decision Tree algorithm specifies the maximum depth of the tree, limiting the number of levels of nodes from the root to the deepest leaf, thereby controlling the complexity of the model and helping prevent overfitting.)

Values considered for hyper-parameters: - None, 5, 10, 15, 20, 30, 40, 50, 60

The best value for the hyper-parameter selected by GridSearchCV: - 5

## 8) Adaboost Classifier

Hyperparameter considered: - classifier\_n\_estimators, classifier\_learning\_rate (In an Adaboost algorithm, the `classifier\_\_n\_estimators` parameter determines the maximum number of weak learners (usually decision trees) to be sequentially added to the ensemble, while the `classifier\_\_learning\_rate` parameter controls the contribution of each weak learner to the final prediction by scaling their weights, influencing the degree of importance each learner has in the model.)

Values considered for hyper-parameters (n\_estimators): - 10,20,30,40,100,70,50,150,200,250, 300

Values considered for hyper-parameters (learning\_rate): - 0.001,0.01,0.1,0.3,1.0

The best value for the hyper-parameter selected by GridSearchCV (n\_estimator): - 50

The best value for the hyper-parameter selected by GridSearchCV(learning rate): - 0.1

## 9) Extra Trees Classifier

Hyperparameter considered: - classifier\_n\_estimators (The `classifier\_\_n\_estimators` parameter in an Extra Trees algorithm determines the number of randomly generated decision trees to be aggregated into the final ensemble model, enhancing robustness and generalization performance through ensemble averaging.)

Values considered for hyper-parameters: - 10,20,30,40,100,70,50,150,200,250, 300

The best value for the hyper-parameter selected by GridSearchCV: - 200

#### 10) **Linear Support Vector Machine Classifier**

Hyperparameter considered:- classifier\_C (The `classifier\_C` parameter in a Linear Support Vector Machines algorithm controls the penalty for misclassification of data points, with smaller values indicating stronger regularization and potentially simpler decision boundaries.)

Values considered for hyper-parameters: - 0.001, 0.01, 100, 1000, 0.1, 1.0, 10.0

The best value for the hyper-parameter selected by GridSearchCV: - 0.1

#### 11) **Stochastic Gradient Descent (SGD) Classifier**

Hyperparameter considered: - loss (The `classifier\_\_loss` parameter in a Stochastic Gradient Descent algorithm specifies the loss function to be minimized during the optimization process, with options such as "hinge" for SVM-like loss, "log" for logistic regression, and others, influencing the model's behavior and performance.)

Values considered for hyper-parameters: - "hinge", "log", "modified\_huber"

The best value for the hyper-parameter selected by GridSearchCV: - log\_loss

#### 12) **Bagging Classifier Classifier**

Hyperparameter considered: - classifier\_n\_estimators (The `classifier\_\_n\_estimators` parameter in a Bagging Classifier algorithm determines the number of base estimators (usually decision trees) to be trained independently and aggregated by averaging or voting, enhancing the model's stability and reducing variance.)

Values considered for hyper-parameters: - 10, 20, 30, 40, 100, 70, 50, 150, 200, 250, 300

The best value for the hyper-parameter selected by GridSearchCV: - 10

## **PERFORMANCE ANALYSIS OF ALL CLASSIFIERS**

In this project, various machine learning algorithms like Random Forest, Gradient Boosting, Support Vector Machines (SVM), Logistic Regression, XGBoost, Naive Bayes, K-Nearest Neighbours (KNN), Decision Tree, Adaboost, Gaussian Naive Bayes, Extra Trees, Linear Support Vector Machines, Stochastic Gradient Decent (SGD), Bagging Classifier are used to predict novel human proteins getting involved in the dengue-human PPI pathway. Various attributes of the protein like PCP\_PC, PCP\_NC, PCP\_NE, PCP\_PO, PCP\_NP, PCP\_AL, PCP\_CY, PCP\_AR, PCP\_AC, PCP\_BS, PCP\_NE\_pH, PCP\_HB, PCP\_HL, PCP\_NT, PCP\_HX, PCP\_SC, PCP\_TN, PCP\_SM, PCP\_LR, PCP\_Z1, PCP\_Z2, PCP\_Z3 etc are considered for this project. The accuracy for individual algorithms has to measure and whichever algorithm is giving the best accuracy, that is considered for the prediction of novel human proteins from the Uniprot [9] database. For evaluating the experiment, various evaluation metrics like accuracy, precision, recall, and f1-score are considered.

TP: True positive (were positive and classified as positive)

FP: False Positive (were negative and classified as positive)

FN: False Negative (were positive and classified as negative)

TN: True Negative (were negative and classifies as negative)

**Correlation Matrix:** The correlation matrix in machine learning is used for feature selection. It represents dependency between various attributes.

A heatmap in data visualization is a powerful tool that visually represents data in a matrix using a color-coded scheme. Each cell in the matrix corresponds to a specific data point, and the color intensity reflects the numerical values. Heatmaps are extensively employed in machine learning

to explore relationships between variables, aiding in the identification of correlations and facilitating informed decision-making.

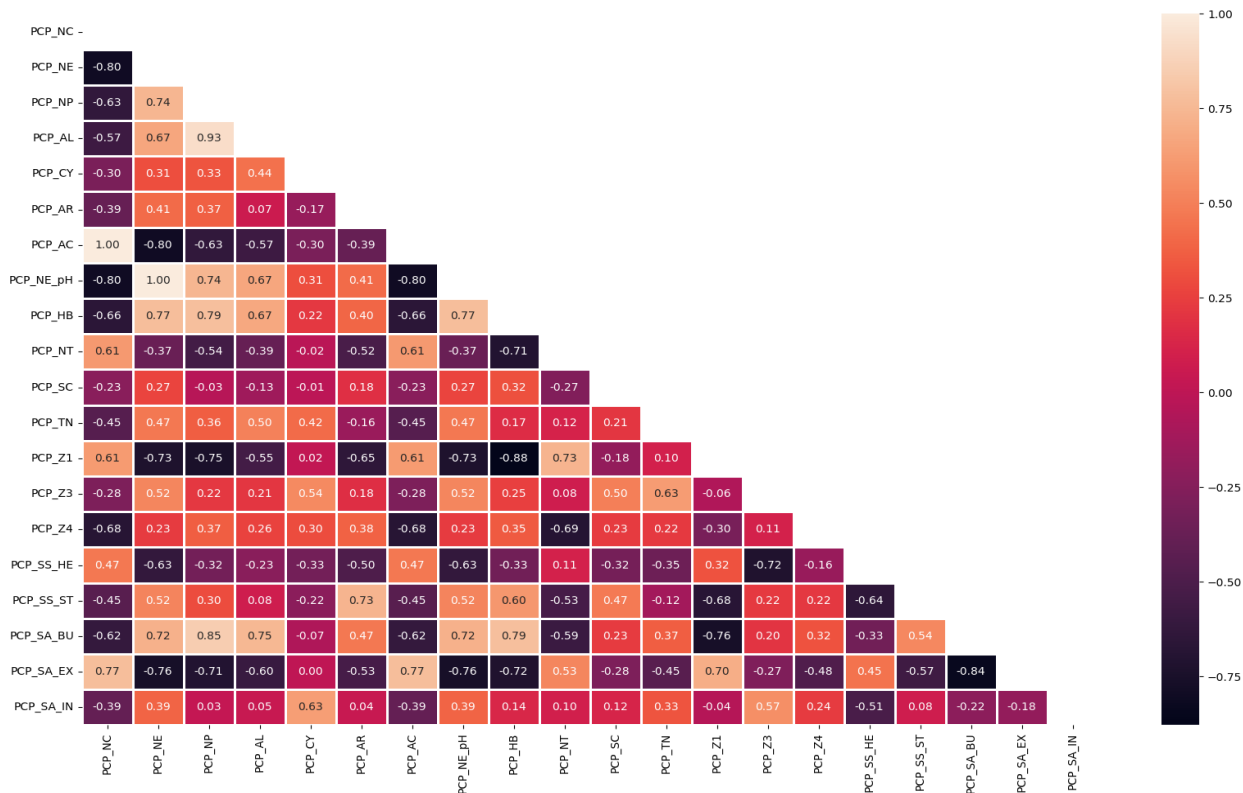


FIGURE 25: - HEAT MAP FOR THE GIVEN DATASET

### Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification algorithm. It displays the actual versus predicted classifications across four categories: true positives (correctly predicted positives), true negatives (correctly predicted negatives), false positives (incorrectly predicted positives), and false negatives (incorrectly predicted negatives). This matrix helps in understanding the accuracy, precision, recall, and overall effectiveness of the model. By analyzing the confusion matrix, one can identify where the model is making errors

and which classes are being confused, thereby providing insights for improving the model's performance.

**Accuracy-** Accuracy is the ratio of the number of correct predictions to the total number of inputs in the dataset. It is expressed as:  $\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{FP} + \text{FN} + \text{TN})$

Accuracy in machine learning is a measure of how well a model correctly predicts both positive and negative instances. It's calculated by dividing the number of correct predictions (true positives and true negatives) by the total number of predictions. A high accuracy indicates a well-performing model, but it may not be sufficient for imbalanced datasets, where one class significantly outnumbers the other. In such cases, additional metrics like precision, recall, or F1 score might provide a more comprehensive evaluation of the model's performance.

**Precision-** It is the ratio of correct positive results to the total number of positive results predicted by the system.  $\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$

Precision is a key metric in machine learning that gauges the accuracy of positive predictions made by a model. It is calculated by dividing the number of true positives by the sum of true positives and false positives. Precision reflects the model's ability to avoid false positives, providing insight into how trustworthy its positive predictions are. In scenarios where minimizing false positives is critical, such as in spam email filters, precision is a vital performance measure.

**Recall-** It is the ratio of correct positive results to the total number of positive results predicted by the system.  $\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$

Recall also known as sensitivity or true positive rate, is a crucial metric in machine learning evaluation. It measures the proportion of actual positive instances correctly identified by a

model. The recall score is calculated by dividing the number of true positives by the sum of true positives and false negatives. A high recall indicates that the model is effective at capturing most positive instances, making it valuable in scenarios where missing positive cases is a significant concern, such as in medical diagnoses or fraud detection.

F1 Score- It is the harmonic mean of Precision and Recall. It measures the test accuracy. The range of this metric is 0 to 1. 
$$F1\ Score = 2 * ((Precision * Recall) / (Precision + Recall))$$

F1 score is a comprehensive metric in machine learning that balances precision and recall. It is the harmonic mean of precision and recall and is particularly useful when there is an uneven class distribution. The F1 score takes into account both false positives and false negatives, providing a single value that represents a trade-off between precision and recall. A high F1 score indicates a model that effectively balances precision and recall, making it suitable for tasks where achieving a balance between false positives and false negatives is crucial.

The Area Under the ROC Curve (AUC)- is a metric in machine learning that assesses the performance of a binary classification model.

The ROC (Receiver Operating Characteristic) curve illustrates the trade-off between true positive rate and false positive rate at various classification thresholds. The AUC quantifies the entire two-dimensional area beneath this curve. A higher AUC value, closer to 1, indicates a better-performing model in terms of distinguishing between positive and negative instances. It serves as a comprehensive measure of a model's ability to discriminate between classes across different decision boundaries.

## Performance Evaluation of the classifiers

There are in total of 30 features that we have considered while preparing this dataset. All these 30 features define the Physico-chemical properties of the individual proteins. After cleaning the dataset, we use SelectKBest features module to find the top 20 features in the dataset and test the correlation coefficient amongst all the possible pairs of 20 features and drop the features which have a correlation coefficient more than 0.75. We are left with 12 columns for the final data frame to be used for training the machine learning models. Dimensionality reduction helps in efficient pattern recognition by the models and decreases computational cost.

We split the dataset in a ratio of 8:2 (train: test) and used it to train Multiple models on it. Used GridSearchCV to tune hyperparameters and find the best possible configuration of the model for the dataset.

Table 3: - Confusion Matrix Values for Individual Classifiers

Classifier	True Positive	False Positive	False Negative	True Negative
Random Forest	43	18	13	47
Gradient Boosting	38	23	12	48
SVM	42	19	16	44
Logistic Regression	40	21	16	44
XGBoost	35	26	17	43
Naive Baye's	38	23	12	48
KNN	31	30	14	46
Decision Tree	24	37	5	55
Adaboost	37	24	14	46
Gaussian Naive Baye's	38	23	12	48
Extra Trees	40	21	12	48
Linear SVM	40	21	15	45
SGD	38	23	29	31
Bagging Classifier	42	19	18	42

Following are the values for the evaluation metrics based on the performance on the test data in descending order: -

	Classifier	Test Score	Precision	Recall	F1-score	ROC Score
10	Extra Trees	0.752066	0.727273	0.800000	0.761905	0.752459
0	Random Forest	0.727273	0.707692	0.766667	0.736000	0.727596
1	Gradient Boosting	0.710744	0.676056	0.800000	0.732824	0.711475
5	Naive Bayes	0.710744	0.676056	0.800000	0.732824	0.711475
9	Gaussian Naive Bayes	0.710744	0.676056	0.800000	0.732824	0.711475
7	Decision Tree	0.652893	0.597826	0.916667	0.723684	0.655055
2	Support Vector Machine	0.710744	0.698413	0.733333	0.715447	0.710929
11	Linear Support Vector Machines	0.702479	0.681818	0.750000	0.714286	0.702869
8	AdaBoost	0.685950	0.657143	0.766667	0.707692	0.686612
3	Logistic Regression	0.694215	0.676923	0.733333	0.704000	0.694536
12	Stochastic Gradient Descent (SGD)	0.644628	0.613333	0.766667	0.681481	0.645628
6	K-Nearest Neighbors (KNN)	0.636364	0.605263	0.766667	0.676471	0.637432
4	XGBoost	0.644628	0.623188	0.716667	0.666667	0.645219
13	Bagging Classifier	0.661157	0.686275	0.583333	0.630631	0.660519

FIGURE 26: METRICS FOR THE MODELS TESTED

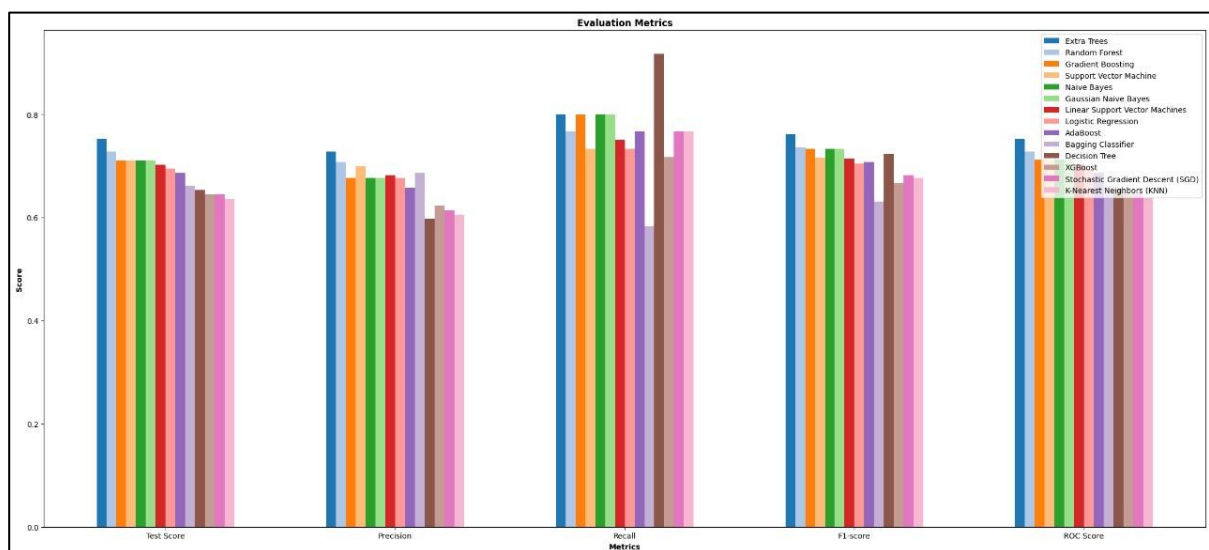


FIGURE 27: MODEL PERFORMANCE COMPARISON USING BAR GRAPHS



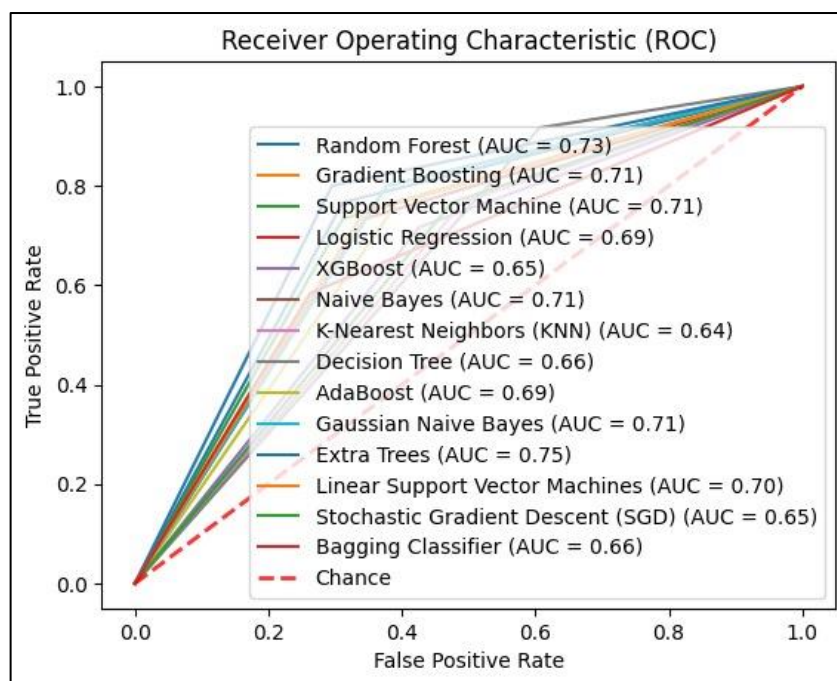


FIGURE 28: AUC VALUES FOR THE MODELS

We select the top 3 models from the list to vote and predict in all human proteins database to narrow down, the proteins that will be selected finally out of the 19834 proteins will be the ones which the resources need to be focused by the researchers, hence saving time and manpower as well as efficient utilization of limited resources.

But before we conduct the voting, we double test the models by letting the top 3 models vote in a list of 34 known positive proteins to see the performance. The list was prepared by conducting literature survey and these proteins were not used in training and testing.

The results for the individual models were as follows: -

Table 4: - Second Phase Testing Results

Model Name	No. of proteins classified as positive	Total no. of positive proteins
Random Forest	15	34
Gradient Boosting	19	34
Extra Trees	16	34
All 3 considered ( $\geq 2$ for +ve)	17	34

If we consider the votes of all the 3 models and set a condition to consider the protein as a positive if it is marked as positive by two or more models. Then the result comes out as 15 out of 34.

Now, we go on to conduct the voting in the database of all human proteins list (except those used in training and testing) using the top 3 models and setting the same condition, i.e. to consider the protein as positive if two or more models denote it as positive. We find that 10343 proteins out of 19834 are being voted as positive.

These 10343 proteins need to be further considered for clinical testing for verification of their role in the dengue-human protein-protein interaction pathway.

## **CHAPTER 6**

# **CONCLUSION**

Multiple novel human proteins were predicted by the top-performing machine learning models, indicating their potential involvement in the dengue-human Protein-Protein Interaction (PPI) pathway. These predictions, if validated, could significantly enhance our understanding of the molecular mechanisms underlying dengue virus infection. The identified proteins could serve as crucial candidates for therapeutic drug targets, offering new avenues for the development of antiviral therapies and potentially revolutionizing the pharmaceutical industry. However, the verification of these predictions is not straightforward and necessitates further investigation. Rigorous validation through molecular docking studies is essential to understand the binding affinities and interactions at the molecular level.

Additionally, extensive clinical trials are required to evaluate the efficacy and safety of targeting these proteins in a therapeutic context. The successful identification and validation of these novel proteins could streamline the development of targeted treatments, leading to more effective management of dengue virus infections. Moreover, this comprehensive list of predicted proteins will facilitate the efficient identification and prioritization of novel proteins for further research, potentially accelerating the pace of discovery and innovation in the field of infectious diseases. Ultimately, this approach underscores the transformative potential of integrating machine learning models with biomedical research. As we continue to uncover these critical interactions, we pave the way for breakthroughs that could mitigate the global health burden of dengue fever.

## **REFERENCES**

- [1](Sessions et al., 2009b) Sessions, O. M., Barrows, N. J., Souza-Neto, J. A., Robinson, T. J., Hershey, C. L., Rodgers, M. A., Ramirez, J. L., Dimopoulos, G., Yang, P. L., Pearson, J. L., & Garcia-Blanco, M. A. (2009b). Discovery of insect and human dengue virus host factors. *Nature*, 458(7241), 1047–1050. <https://doi.org/10.1038/nature07967>
- [2] (Doolittle & Gomez, 2011b) Doolittle, J. M., & Gomez, S. M. (2011b). Mapping Protein Interactions between Dengue Virus and Its Human and Insect Hosts. *PLoS Neglected Tropical Diseases*, 5(2), e954. <https://doi.org/10.1371/journal.pntd.0000954>
- [3] (Le Sommer et al., 2012) Le Sommer, C., Barrows, N. J., Bradrick, S. S., Pearson, J. L., & Garcia-Blanco, M. A. (2012). G Protein-Coupled Receptor Kinase 2 Promotes Flaviviridae Entry and Replication. *PLoS Neglected Tropical Diseases*, 6(9), e1820. <https://doi.org/10.1371/journal.pntd.0001820>
- [4] (Mairiang et al., 2013) Mairiang, D., Zhang, H., Sodja, A., Murali, T., Suriyaphol, P., Malasit, P., Limjindaporn, T., & Finley, R. L. (2013). Identification of New Protein Interactions between Dengue Fever Virus and Its Hosts, Human and Mosquito. *PLoS ONE*, 8(1), e53535. <https://doi.org/10.1371/journal.pone.0053535>
- [5] (Savidis et al., 2016) Savidis, G., McDougall, W. M., Meraner, P., Perreira, J. M., Portmann, J. M., Trincucci, G., John, S. P., Aker, A. M., Renzette, N., Robbins, D. R., Guo, Z., Green, S., Kowalik, T. F., & Brass, A. L. (2016). Identification of Zika Virus and Dengue Virus Dependency Factors using Functional Genomics. *Cell Reports*, 16(1), 232–246. <https://doi.org/10.1016/j.celrep.2016.06.028>
- [6] (Viktorovskaya et al., 2016) Viktorovskaya, O. V., Greco, T. M., Cristea, I. M., & Thompson, S. R. (2016). Identification of RNA Binding Proteins Associated with Dengue Virus RNA in Infected Cells Reveals Temporally Distinct Host Factor Requirements. *PLOS Neglected Tropical Diseases*, 10(8), e0004921. <https://doi.org/10.1371/journal.pntd.0004921>
- [7] (Rothan & Kumar, 2019) Rothan, H. A., & Kumar, M. (2019). Role of Endoplasmic Reticulum-Associated Proteins in Flavivirus Replication and Assembly Complexes. *Pathogens*, 8(3), 148. <https://doi.org/10.3390/pathogens8030148>
- [8] (Farooq et al., 2023) Farooq, Q. ul A., Aiman, S., Ali, Y., Shaukat, Z., Ali, Y., Khan, A., Samad, A., Wadood, A., & Li, C. (2023). A comprehensive protein interaction map and druggability investigation prioritized dengue virus NS1 protein as promising therapeutic candidate. *PLOS ONE*, 18(7), e0287905. <https://doi.org/10.1371/journal.pone.0287905>

- [9] (Bateman et al., 2021) Bateman, A., Martin, M.-J., Orchard, S., Magrane, M., Agivetova, R., Ahmad, S., Alpi, E., Bowler-Barnett, E. H., Britto, R., Bursteinas, B., Bye-A-Jee, H., Coetzee, R., Cukura, A., Da Silva, A., Denny, P., Dogan, T., Ebenezer, T., Fan, J., Castro, L. G., ... Teodoro, D. (2021). UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Research*, 49(D1), D480–D489. <https://doi.org/10.1093/nar/gkaa1100> **Nucleic Acids Res. 49:D480–D489(2021)** ( [www.uniprot.org](http://www.uniprot.org) )
- [10] (Pande et al., 2023) Pande, A., Patiyal, S., Lathwal, A., Arora, C., Kaur, D., Dhall, A., Mishra, G., Kaur, H., Sharma, N., Jain, S., Usmani, S. S., Agrawal, P., Kumar, R., Kumar, V., & Raghava, G. P. S. (2023). Pfeature: A Tool for Computing Wide Range of Protein Features and Building Prediction Models. *Journal of Computational Biology*, 30(2), 204–222. <https://doi.org/10.1089/cmb.2022.0241> ( <https://webs.iiitd.edu.in/raghava/pfeature/> )
- [11] <https://medium.com/@nandiniverma78988/understanding-and-implementing-gaussian-naive-bayes-classification-with-python-dbdcf2939f7>
- [12] <https://medium.com/@datasciencewizards/understanding-the-adaboost-algorithm-2e9344d83d9b>
- [13] <https://medium.com/@datasciencewizards/understanding-the-gradient-boosting-algorithm-9fe698a352ad>
- [14] <https://medium.com/@arch.mo2men/what-is-bagging-classifier-45df6ce9e2a1>
- [15] <https://medium.com/@akhil0435/linear-svm-classification-40dde297c931>
- [16] <https://medium.com/@namanbhandari/extratreesclassifier-8e7fc0502c7>
- [17] <https://www.upgrad.com/blog/gaussian-naive-bayes/>
- [18] (Olivier Grisel et al., 2011) Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, & Matthieu Perrot, E. D. (2011). *Scikit-learn: Machine Learning in Python*. 445, 2825–2830. <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>
- [19] (Hunter, 2007) Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- [20] (Harris et al., 2020) Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>

[21] (McKinney, 2010) McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. 56–61. <https://doi.org/10.25080/Majora-92bf1922-00a>

[22] (Dey & Mukhopadhyay, 2017) Dey, L., & Mukhopadhyay, A. (2017). DenvInt: A database of protein–protein interactions between dengue virus and its hosts. *PLOS Neglected Tropical Diseases*, 11(10), e0005879. <https://doi.org/10.1371/journal.pntd.0005879>

## **APPENDIX**

Link to the GitHub Repository:

<https://github.com/SahaFouzder-Raunak/DENV-Human-PPI-analysis-for-Novel-H-proteins-detection>