# Btech Final Year Project Report

On

# “Indian Loksabha Election Prediction Using Machine Learning: Comparative Analysis of Classification Model”

By

| | |
|---|---|
| Pranav Bhople | 121B1D024 |
| Dnyaneshwari Katkar | 121B1D049 |
| Sahil Dhanwani | 121B1D052 |
| Vaibhav Indure | 121B1D053 |

Under the guidance of

**Prof. Anandkumar Birajdar**

**DEPARTMENT OF COMPUTER ENGINEERING(RL)**
**PCET’S PIMPRI CHINCHWAD COLLEGE OF ENGINEERING**
**Sector No. 26, Pradhikaran, Nigdi,**
**Pune - 411044**

# CERTIFICATE

This is to certify that the Final Year B-Tech project report entitled

**"Indian Loksabha Election Prediction Using Machine Learning: Comparative Analysis of Classification Models"**

has been successfully completed and submitted by the following students:

**Pranav Bhople (121B1D024)**
**Dnyaneshwari Katkar (121B1D049)**
**Sahil Dhanwani (121B1D052)**
**Vaibhav Indure (121B1D053)**

The report is approved by Prof. Anandkumar Birajdar for submission. It is certified further that, to the best of my knowledge, this report represents original work carried out by these students as partial fulfillment of the Final Year B. Tech in Computer Engineering (Regional Language, Semester 8), as per the curriculum of PCET's Pimpri Chinchwad College of Engineering, Nigdi, Pune for the academic year 2024–25.

| | |
|---|---|
| **Prof. Anandkumar Birajdar** | **Prof. Dr. Rachana Y. Patil** |
| Project Guide | Head of Department |

**Date:** 8/4/2025 **Place:** Pimpri Chinchwad College of Engineering, Pune

## Acknowledgment

We express our sincere thanks to our Guide **Prof. Anandkumar Birajdar** for his constant encouragement and support throughout our project, especially for the useful suggestions given during the course of project and having laid down the foundation for the success of this work.

We would also like to thank our Project Coordinator, **Prof. Rohini Sarode** for her assistance, genuine support and guidance from early stages of the project. We would like to thank **Prof. Dr. Rachana Y. Patil**, Head of Computer Department (Regional Language) for her unwavering support during the entire course of this project work. We are very grateful to our Director, **Prof. Dr. G.N. Kulkarni** for providing us with an environment to complete our project successfully. We also thank all the staff members of our college and technicians for their help in making this project a success. We also thank all the web committees for enriching us with their immense knowledge. Finally, we take this opportunity to extend our deep appreciation to our family and friends, for all that they meant to us during the crucial times of the completion of our project.

| **Name of the Student** | **PRN** |
|---|---|
| Pranav Bhople | 121B1D024 |
| Dnyaneshwari Katkar | 121B1D049 |
| Sahil Dhanwani | 121B1D052 |
| Vaibhav Indure | 121B1D053 |

# Abstract

This study presents a machine learning-based approach to predict election outcomes using real-world data from the Indian Lok Sabha Elections 2024. We assess the significance of various candidate and constituency attributes—including party affiliation, age, total votes, total electors, assets, liabilities, education level, criminal cases, gender, and category—through XGBoost's feature importance analysis.

To build a robust predictive model, we evaluate ten machine learning algorithms: Random Forest, SVM, KNN, Logistic Regression, XGBoost, CatBoost, Gradient Boosting, AdaBoost, LightGBM, and Decision Tree, using 10-Fold Cross-Validation to minimize overfitting. The models are then compared based on accuracy and other evaluation metrics to determine the most effective approach for election forecasting. Our findings contribute to the field of political data analytics by demonstrating the potential of machine learning in predicting electoral trends with high accuracy.

**Key words:** Election Prediction, Machine Learning, Political Data Analytics, Feature Selection, XGBoost, 10-Fold Cross-Validation, Classification Algorithms, Model Evaluation, Electoral Forecasting.

# Contents

# Abbreviations

| Abbreviation | Full Form |
|---|---|
| AUC | Area Under the Curve |
| CV | Cross-Validation |
| EDA | Exploratory Data Analysis |
| KNN | K-Nearest Neighbors |
| ROC | Receiver Operating Characteristic |
| SMOTE | Synthetic Minority Over-sampling Technique |
| SVM | Support Vector Machine |
| XGBoost | eXtreme Gradient Boosting |

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Overview

Elections are central to democratic governance, shaping national policies and leadership. Traditional methods for election prediction, such as opinion polls and surveys, often suffer from biases and limited accuracy. In contrast, machine learning techniques offer a data-driven, objective approach to forecasting electoral outcomes by analyzing historical and real-time data. This research aims to leverage advanced machine learning models to predict election outcomes with greater accuracy and reliability.

## 1.2 Motivation

Predicting election results accurately is crucial for political analysts, policymakers, and the general public. Conventional prediction methods often fail to account for dynamic political trends and hidden correlations within electoral data. Machine learning models provide an opportunity to analyze large datasets efficiently, extract meaningful insights, and enhance the transparency of the electoral process. By incorporating features such as candidate attributes, voter demographics, and historical election trends, we aim to develop a robust predictive framework.

## 1.3 Problem Statement and Objectives

Traditional election prediction techniques rely heavily on manual surveys and statistical analyses, which often introduce biases and inaccuracies. The primary challenge is to develop a model that can effectively learn from past election data and generalize well to future elections.

Objectives of this research include:

- Developing a machine learning-based election prediction model.
- Identifying key candidate and constituency attributes that influence election outcomes.
- Evaluating and comparing multiple machine learning algorithms to determine the most accurate model.
- Validating predictions using real-world election data and performance metrics.
- Providing interpretability in model outcomes to understand feature importance in election results.

## 1.4 Scope of the Work

This research focuses on predicting the outcomes of the Indian Lok Sabha Elections 2024 using machine learning techniques. The study encompasses:

- Data collection from past election results, candidate profiles, and constituency-level statistics.
- Feature engineering to extract meaningful patterns from structured and unstructured data.
- Implementation of ten machine learning models, including ensemble methods.
- Performance evaluation using accuracy, precision, recall, F1-score, and computational efficiency.
- Interpretation of feature importance and model explainability using visualization techniques.

## 1.5 Methodologies of Problem Solving

To develop an effective election prediction model, we adopt the following structured methodology:

- **Data Collection**: Gathering historical election data, candidate attributes, and constituency demographics from credible sources.

- **Data Preprocessing**: Handling missing values, encoding categorical variables, and normalizing numerical features.
- **Feature Selection**: Identifying key attributes such as party affiliation, assets, liabilities, education, and criminal records.
- **Model Training**: Implementing various machine learning models, including CatBoost, XGBoost, LightGBM, Random Forest, Logistic Regression, and SVM.
- **Hyperparameter Tuning**: Optimizing model parameters to achieve the best performance using techniques like Grid Search and Random Search.
- **Evaluation**: Assessing models using 10-Fold Cross-Validation and performance metrics.
- **Validation and Interpretation**: Analyzing feature contributions and visualizing predictions through confusion matrices and ROC curves.

The outcomes of this research aim to enhance the accuracy of election forecasting, providing valuable insights for political strategists and researchers.

# Chapter 2

# Literature Survey

## 2.1 Review of recent literature

Predicting election outcomes through machine learning has become a widely explored area in recent years, primarily due to the growing use of data-driven approaches to enhance prediction accuracy. Many recent works investigate diverse machine learning models such as Random Forest, Neural Networks, and ensemble approaches to study voter behavior and forecast results. These investigations emphasize the critical role of feature selection, data preprocessing, and model tuning in boosting prediction accuracy. Despite these advancements, issues like data bias and shifting political trends continue to pose challenges. This literature review aims to assess recent progress, highlight key methodologies, and identify gaps in current research to inform the present study.

A review of selected academic studies is outlined below:

[1] Singh, Harmanjeet, and Anand Kumar Shukla (2021) examine how machine learning algorithms can support the analysis and forecasting of Indian election results. The study employs supervised learning models to process election data and determine major factors that influence voting behavior. Utilizing historical election records, the research develops forecasting models designed to provide reliable predictions. The findings showcase the value of machine learning in electoral studies, especially within India's diverse and dynamic political environment, where conventional forecasting methods may not suffice.

[2] Amit Kumar Yadav and Rahul Johari (2021) focus their study on pattern recognition and prediction of Indian electoral outcomes using historical data and machine learning techniques. By investigating past election results, the authors apply machine learning algorithms to discover hidden correlations in the data and to project future voter behavior. Their work underlines the

importance of feature selection and thorough data preprocessing in building dependable predictive models.

[3] The study by Abhay Singh Bhadauria and colleagues (2024) offers a comparative analysis between deep learning methods and conventional machine learning models in predicting election sentiments. By utilizing social media and other online data sources, the researchers assessed voter opinions and public sentiment to predict election trends. The results showed that deep learning approaches, particularly neural networks, achieved better accuracy in sentiment analysis than traditional models like decision trees or logistic regression. This research highlights the increasing relevance of deep learning in political forecasting, especially for analyzing unstructured and sentiment-heavy data sources.

[4] Gaurav Mandhyan and Shreea Bose (2024) conducted a study that explores how machine learning can be used to forecast results in Indian general elections. Their analysis involved various datasets including demographic information, previous election records, and socio-political indicators to build predictive models. Several algorithms were tested to assess their performance in terms of accuracy and dependability, particularly within India's complex, multi-party electoral system. The findings suggest that machine learning is capable of capturing nuanced electoral dynamics, thereby serving as a helpful tool for political analysts and policymakers.

5] Zuloaga-Rotta et al. (2024) proposed a hybrid model that combines simulation-based techniques with machine learning for predicting presidential election outcomes. Their approach integrates probabilistic simulations with prediction algorithms to enhance the credibility of electoral forecasts. By merging diverse datasets—such as demographic statistics, polling results, and historical election trends—the model aims to provide realistic projections of voting behavior. The study concludes that this hybrid strategy improves forecast accuracy and can be effectively adapted for use in different national election systems.

## 2.2 Gap Identification / Common findings from literature

Recent studies on election prediction using machine learning highlight key aspects of model performance, feature selection, and data sources:

### 2.2.1 Common Findings from the Literature

- All studies affirm that machine learning models can successfully analyze and predict election outcomes using historical, demographic, or sentiment-based data.
- Algorithms like Random Forest, SVM, Logistic Regression, and Neural Networks were commonly employed, with varying levels of success.
- Each paper emphasized that data preprocessing, feature selection, and data cleaning are essential steps in building reliable models.
- Quality of data—such as completeness, consistency, and relevance—was directly tied to the accuracy of predictions.
- Bhadauria et al. and Zuloaga-Rotta et al. found that deep learning models like neural networks provided superior performance in sentiment-based and simulation-based forecasting.
- However, traditional models still performed well in structured and historical data contexts (e.g., Mandhyan & Bose, Yadav & Johari).
- The diversity in data sources improved model robustness and prediction accuracy.

### 2.2.2 Research Gaps Identified

- Most models used historical election data only, without incorporating real-time data like current voter sentiment, ongoing political campaigns, or live polling data.
- Elections are dynamic — public opinion can shift due to recent events. Real-time updates can improve prediction accuracy.
- Only Bhadauria et al. (2024) used social media for sentiment analysis. Others missed this rich, real-time source of voter opinion.
- Few papers compare a broad range of ML and deep learning models on the same dataset. Most focus on just one or two algorithms.
- Datasets used in Indian election prediction often focus on specific states or general elections without granular local-level data.

- India is highly diverse. Predictive accuracy would improve with more localized, demographically segmented data.
- Only one paper used a hybrid model combining simulation techniques with machine learning.
- Deep learning models are often treated as black boxes, without insights into why they predict a certain outcome.
- No comparative analysis of election prediction models across different countries or electoral systems.
- Benchmarking Indian election forecasting techniques against models used in other democracies could reveal better strategies or universal trends.

# Chapter 3

# Software Requirements Specification

## 3.1 Functional Requirements

### 3.1.1 System Feature 1: Data Preprocessing (Exploratory Data Analysis - EDA)

This feature involves preprocessing the dataset, handling missing values, encoding categorical variables, and performing exploratory data analysis (EDA). The key steps include:

- Data cleaning, handling missing values, and removing inconsistencies.
- Encoding categorical variables using techniques such as Label Encoding and One-Hot Encoding.
- Feature scaling using MinMaxScaler to normalize data.
- Visualizing data distributions using seaborn, matplotlib, and plotly.
- Identifying data imbalances and applying techniques such as SMOTE for resampling.
- Splitting the dataset into training and testing subsets for further processing.

### 3.1.2 System Feature 2: Result Comparison of 10 Algorithms and Finding the Best One

This feature involves training multiple machine learning models, evaluating their performance, and selecting the best algorithm based on accuracy and other metrics. The models include:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Gradient Boosting Classifier
- AdaBoost Classifier
- Support Vector Classifier (SVC)
- K-Nearest Neighbors (KNN)
- XGBoost Classifier
- CatBoost Classifier
- LightGBM Classifier

The evaluation metrics include:

- Accuracy Score
- Precision, Recall, and F1-Score
- Confusion Matrix Analysis
- ROC Curve and AUC Score
- Cross-Validation for Model Generalization

The best model will be selected based on overall performance and suitability for the dataset.

## 3.2 External Interface Requirements

This section describes the external interfaces required for data collection and model evaluation. The primary sources of data include:

- Government websites providing publicly available datasets related to elections.
- Historical election data from official records and research sources.

- Web scraping techniques, if applicable, to collect relevant election-related data.

The collected data will be stored in structured formats (CSV, JSON, or databases) for further analysis and preprocessing.

## 3.3 System Features

### 3.3.1 User Authentication and Authorization

- The system shall allow users to register, log in, and log out securely. - Authentication shall be handled using secure protocols. - Role-based access control shall be implemented.

### 3.3.2 Data Processing and Analysis

- The system shall handle large datasets using efficient processing techniques. - Data visualization shall be integrated using Plotly and Seaborn. - Statistical computations shall utilize NumPy and Pandas.

## 3.4 Performance Requirements

- The system shall provide real-time feedback on user actions. - Response time shall not exceed 2 seconds for any major operation. - Machine learning models shall be optimized for efficiency.

## 3.5 Design Constraints

- The system shall be developed using Python and associated libraries such as NumPy, Pandas, Scikit-learn, and Plotly. - Visualization tools such as Matplotlib, Seaborn, and Plotly shall be used. - Data resampling shall be handled using SMOTE to balance datasets.

## 3.6 Software System Attributes

### 3.6.1 Security

- The system shall encrypt user credentials using industry-standard encryption methods. - Access to machine learning models shall be restricted based on user roles.

### 3.6.2 Scalability

- The system shall support increasing datasets and growing user interactions without performance degradation. - Cloud-based storage solutions shall be considered for scalability.

### 3.6.3 Usability

- The system shall have an intuitive user interface with clear visualization tools. - Users shall be able to interact with data using interactive plots.

## 3.7 Dependencies and Assumptions

- The system assumes users have access to an internet connection. - Required Python libraries and frameworks shall be installed before execution. - The machine learning models depend on Scikit-learn, XGBoost, CatBoost, and LightGBM for execution.

# Chapter 4

# System Design

## 4.1 Proposed System Architecture

The proposed system is a machine learning-based election prediction model that analyzes real-world data from the Indian Lok Sabha Elections 2024. The architecture follows a structured workflow comprising multiple stages, including:

- **Data Collection** – Aggregating election-related data from sources like the Election Commission of India (ECI) and myneta.com.
- **Data Preprocessing** – Cleaning, encoding, and transforming the dataset for consistency and efficiency.
- **Exploratory Data Analysis (EDA)** – Understanding data patterns through statistical and visual analysis.
- **Feature Selection** – Identifying the most relevant attributes using XGBoost feature importance analysis.
- **Model Training & Selection** – Evaluating multiple machine learning algorithms to determine the best-performing model.
- **Evaluation & Prediction** – Optimizing the final model and making predictions on election outcomes.

A flow diagram representing the above architecture is depicted below:

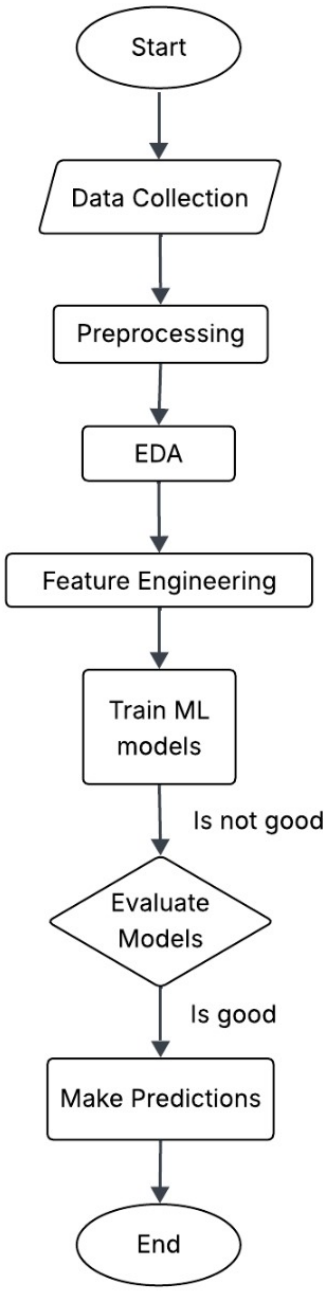


Figure 4.1: Proposed System Architecture

## 4.2 Dataset design

The dataset consists of various features related to candidates, constituencies, and election results. The attributes include:

- **Candidate Details**: Name, age, gender, education, criminal cases, assets, liabilities.
- **Constituency Information**: State, constituency, total electors.
- **Election Outcomes**: Party affiliation, total votes, winner status.

The database schema follows a relational model, where candidates, constituencies, and election results are stored in single table for efficient querying.

## 4.3 Entity Relationship Diagrams (ERD)

To efficiently manage election-related data, we designed an Entity-Relationship Diagram (ERD) that captures the relationships between candidates, constituencies, and election results. The ERD consists of three main entities:

### 4.3.1 Entities and Attributes

**1. Constituency**

Represents the electoral region where candidates contest elections.

- **Constituency Name** – Name of the constituency.
- **State** – The state to which the constituency belongs.
- **Total Electors** – The total number of registered voters in the constituency.

**2. Candidate**

Represents an individual contesting in an election.

- **Candidate Name** – Name of the candidate.
- **Age** – Age of the candidate.
- **Gender** – Gender of the candidate.

- **Education** – Educational qualification.
- **Category** – Caste or reservation category (General, SC, ST, OBC, etc.).
- **Assets** – Declared financial assets.
- **Liabilities** – Declared financial liabilities.
- **Criminal Cases** – Number of criminal cases (if any).
- **Party Name** – Name of the political party the candidate represents.
- **Party Symbol** – Symbol of the political party.

**3. Election Result**

Stores information about the election results for each candidate and constituency.

- **Candidate Name** (Derived from Candidate entity) – Name of the candidate contesting in the election.
- **Constituency** (Derived from Constituency entity) – Name of the constituency where the candidate contested.
- **Winner Status** – Indicates whether the candidate won or lost the election.
- **Total Votes Received** – Number of votes received by the candidate.
- **Total Voting** – Total number of votes cast in the constituency.
- **Percentage Over Total Electors in Constituency** – Percentage of votes received relative to the total number of electors.
- **Percentage Over Total Votes Polled in Constituency** – Percentage of votes received relative to the total votes cast.

### 4.3.2 Relationships

**1. Constituency Has Candidate**

- A constituency has multiple candidates contesting the election.
- Each candidate is uniquely associated with one constituency.

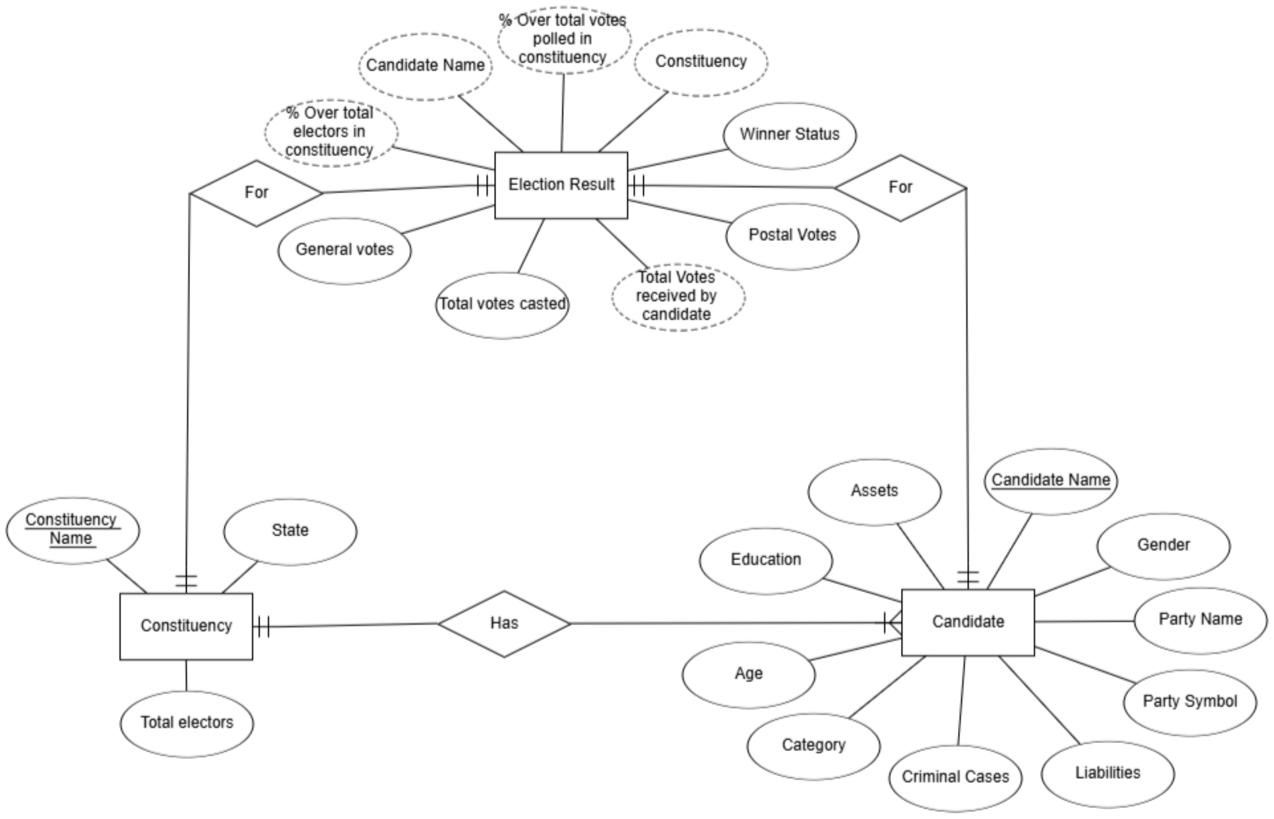


Figure 4.2: ER Diagram

**2. Election Result For Candidate**

- An election result is associated with a specific candidate.
- It includes additional details like postal votes, general votes, and winner status.

**3. Election Result For Constituency**

- Each election result is linked to a specific constituency.
- The results reflect the voting outcome for all candidates in that constituency.

## 4.4 Mathematical Model

The prediction problem is framed as a binary classification task, where:

$$Y = f(X) + \epsilon$$

where:

- $Y$ is the predicted election outcome (Winner = 1, Non-Winner = 0).
- $X$ represents input features (party, votes, assets, liabilities, etc.).
- $f$ is the machine learning model trained on election data.
- $\epsilon$ is the error term representing uncertainty.

The model optimization minimizes the log loss function, defined as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where $y_i$ is the actual outcome and $\hat{y}_i$ is the predicted probability.

# Chapter 5

# Project Plan

## 5.1 Project Cost Estimate (If applicable)

The Election Prediction project relies primarily on open-source tools such as Python, scikit-learn, and XGBoost. Thus, the estimated cost is minimal. However, costs may arise for the following:

- Data acquisition (e.g., Election Commission of India records)
- Software licensing (if premium services are utilized)
- Miscellaneous expenses such as storage, internet charges, etc.

The estimated budget for the project is approximately INR 5,000 to INR 10,000, accounting for cloud resources and data acquisition.

## 5.2 Risk Management

### 5.2.1 Risk Identification

Potential risks identified for this project include:

- Data availability and quality issues
- Model overfitting or underfitting
- Deployment challenges on cloud platforms

- Timeline delays due to unforeseen issues
- Security concerns while handling sensitive data

### 5.2.2 Risk Analysis

Each identified risk is analyzed below:

- **Data Quality:** Inconsistent or noisy data may impact model accuracy. Mitigation includes rigorous data cleaning and preprocessing.
- **Model Performance:** Overfitting may occur if the model is overly complex. Cross-validation and proper hyperparameter tuning will mitigate this.
- **Deployment Issues:** Errors may arise during deployment on Azure. Comprehensive testing before deployment will reduce this risk.
- **Timeline Delays:** Regular progress reviews and milestones will ensure the project stays on track.

### 5.2.3 Overview of Risk Mitigation, Monitoring, and Management

To manage risks effectively:

- Weekly meetings will review progress and potential risks.
- Version control (e.g., Git) will track code changes and enable quick rollbacks if needed.
- Frequent model evaluation will ensure performance remains optimal.

## 5.3 Project Schedule

### 5.3.1 Project Task Set

The project is divided into the following key tasks:

1. Data Collection and Cleaning
2. Exploratory Data Analysis (EDA)

3. Feature Engineering
4. Model Building and Training
5. Model Evaluation and Testing
6. Deployment on Azure
7. Report Writing and Documentation

### 5.3.2 Timeline Chart

The following timeline outlines the expected completion schedule for each task:

| **Task** | **Duration** |
|---|---|
| Data Collection and Cleaning | Week 1 – Week 2 |
| EDA and Feature Engineering | Week 3 – Week 4 |
| Model Building and Evaluation | Week 5 – Week 7 |
| Deployment | Week 8 |
| Report Writing and Finalization | Week 9 – Week 10 |

Table 5.1: Project Timeline with Task Duration

## 5.4 Implementation/Proof of Concept

The project implementation involves:

- Developing machine learning models using Python libraries such as scikit-learn and XGBoost.
- Visualizing data trends and model performance using Streamlit.

## 5.5 Team Organization (Structure)

The project team is organized as follows:

- **Project Lead:** Responsible for overall project coordination and progress tracking.

- **Data Scientist:** Focuses on data collection, cleaning, and model development.
- **Documentation Specialist:** Prepares the final report and ensures documentation is clear and complete.

# Chapter 6

# Project Implementation

This chapter details the implementation of the election prediction system, including the modular breakdown of the project, the tools and technologies used, and the core algorithms employed. The implementation follows a systematic approach, from data acquisition to model deployment, ensuring accuracy and efficiency in predicting election outcomes.

## 6.1 Overview of Project Modules

The project is structured into multiple modules, each focusing on a specific aspect of data processing, analysis, and prediction. The key modules are:

### 6.1.1 Data Collection

- Responsible for gathering real-world election data from multiple sources, including the Election Commission of India (ECI) and MyNeta.com.
- The data includes details about candidates, constituencies, parties, demographics, and past election results.

### 6.1.2 Data Preprocessing

- Cleans and standardizes the collected data by handling missing values, encoding categorical variables, and scaling numeric features.

- Implements Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance issues.

### 6.1.3 Exploratory Data Analysis (EDA)

- Analyzes trends, distributions, and correlations within the dataset.
- Visualizes key attributes such as age distribution, winning probability by party, caste representation, and criminal records analysis.

### 6.1.4 Feature Selection

- Identifies the most important attributes influencing election outcomes using XGBoost's feature importance analysis.

### 6.1.5 Model Training and Evaluation

- Trains multiple machine learning models, including Random Forest, SVM, KNN, Logistic Regression, XGBoost, CatBoost, Gradient Boosting, AdaBoost, LightGBM, and Decision Tree.
- Uses 10-Fold Cross-Validation to enhance model generalization.
- Compares models based on performance metrics such as accuracy, precision, recall, and F1-score.

### 6.1.6 Model Selection and Prediction

- Selects the best-performing model for final prediction.
- Outputs the winning probability of candidates based on historical and current election data.

## 6.2 Tools and Technologies Used

To implement the system efficiently, we employed various tools and technologies, categorized as follows:

### 6.2.1 Programming Languages

- **Python** – The primary language used for data processing, model training, and evaluation.

### 6.2.2 Libraries and Frameworks

- **NumPy & Pandas** – For data manipulation and preprocessing.
- **Matplotlib & Seaborn** – For data visualization in Exploratory Data Analysis (EDA).
- **Scikit-Learn** – For implementing machine learning models and evaluation metrics.
- **XGBoost & CatBoost** – For advanced gradient boosting-based classification models.
- **SMOTE (Imbalanced-learn library)** – For handling class imbalance in the dataset.

### 6.2.3 Data Sources

- **Election Commission of India (ECI)** – Official election data provider.
- **MyNeta.com** – Provides candidate details, including criminal records, assets, and liabilities.

### 6.2.4 Tools for Model Deployment

- **VS Code & Jupyter Notebook** – For model development and experimentation.

### 6.2.5 Version Control & Collaboration

- **Git & GitHub** – For managing code and collaborating with team members.

## 6.3 Data Collection Module

The Data Collection Module is a crucial component of the project, as it lays the foundation for predictive modeling by gathering relevant election-related data. Since real-world election data is spread across multiple sources, it was necessary to collect, clean, and merge it systematically. The data was retrieved from three different tables obtained from various online sources, including MyNeta.com and the Election Commission of India (ECI) website.

### 6.3.1 Data Sources and Extraction Methods

The data was collected from the following sources:

#### Candidate Details from MyNeta.com

The MyNeta.com website provides detailed profiles of candidates contesting elections, including:

- Candidate Name
- Age
- Education Level
- Criminal Record (number of pending cases)
- Assets (declared movable and immovable property)
- Liabilities (loans, unpaid dues, etc.)
- Political Party Affiliation

Since MyNeta.com does not provide direct download options, web scraping techniques were employed to extract this data. Python libraries such as **BeautifulSoup** and **Selenium** were used to automate data extraction, ensuring that all relevant candidate details were retrieved efficiently.

#### Constituency Details from the Election Commission of India (ECI) Website

The ECI website provides official election-related data, including:

- Total number of electors (voters) in each constituency
- Constituency Name
- State Name

This data was crucial for understanding the voter distribution across different regions and for analyzing how candidate attributes relate to election outcomes. Unlike MyNeta.com, some of this data was directly available for download, while other parts required additional processing before integration.

**Election Results from the Election Commission of India (ECI) Website**

The final election results were obtained from the ECI website, which included:

- Candidate Name
- Party Affiliation
- Total Votes Received
- Winner Status (Won/Lost)
- Percentage of Votes Secured

This data helped in determining the factors that influence election victory, forming the basis for model training and evaluation.

### 6.3.2 Data Merging and Storage

After collecting data from the three different sources, the next step was to merge them into a unified dataset. Since each dataset had a different structure, the following steps were taken:

**Data Cleaning & Standardization**

- Candidate names from MyNeta.com and ECI election results were matched to ensure consistency.
- Categorical values (such as party names) were standardized to maintain uniformity.

**Data Merging**

- The tables were merged using the candidate name and constituency as common keys.
- **Pandas** merge operations in Python were used to combine multiple tables into a single comprehensive dataset.

**Final Data Storage**

- The processed data was stored in CSV format for easy accessibility and future analysis.
- The dataset was now ready for the next stage: **Data Preprocessing**.

## 6.4 Data Preprocessing Module

To ensure high-quality input data for machine learning models, the following preprocessing techniques were applied:

- **Handling Missing Data**: Records with incomplete critical values were removed.
- **Cleaning Categorical Variables**: Standardizing inconsistent text data.
- **Numeric Data Transformation**: Converting financial attributes to numeric formats.
- **Feature Encoding**: Using Ordinal Encoding for categorical variables.
- **Feature Scaling**: Applying Min-Max Scaling for normalization.
- **Handling Class Imbalance**: Using SMOTE (Synthetic Minority Over-sampling Technique) to balance dataset distribution.

## 6.5 Exploratory Data Analysis (EDA) Module

Before selecting features and training models, we conducted a detailed Exploratory Data Analysis (EDA) to understand the statistical relationships within the dataset. Various visualizations and statistical techniques were employed to analyze trends, distributions, and correlations among different features.

Key insights from EDA:

- **Winning Probability by Party:** The proportion of winning candidates per party was analyzed to observe dominance trends.

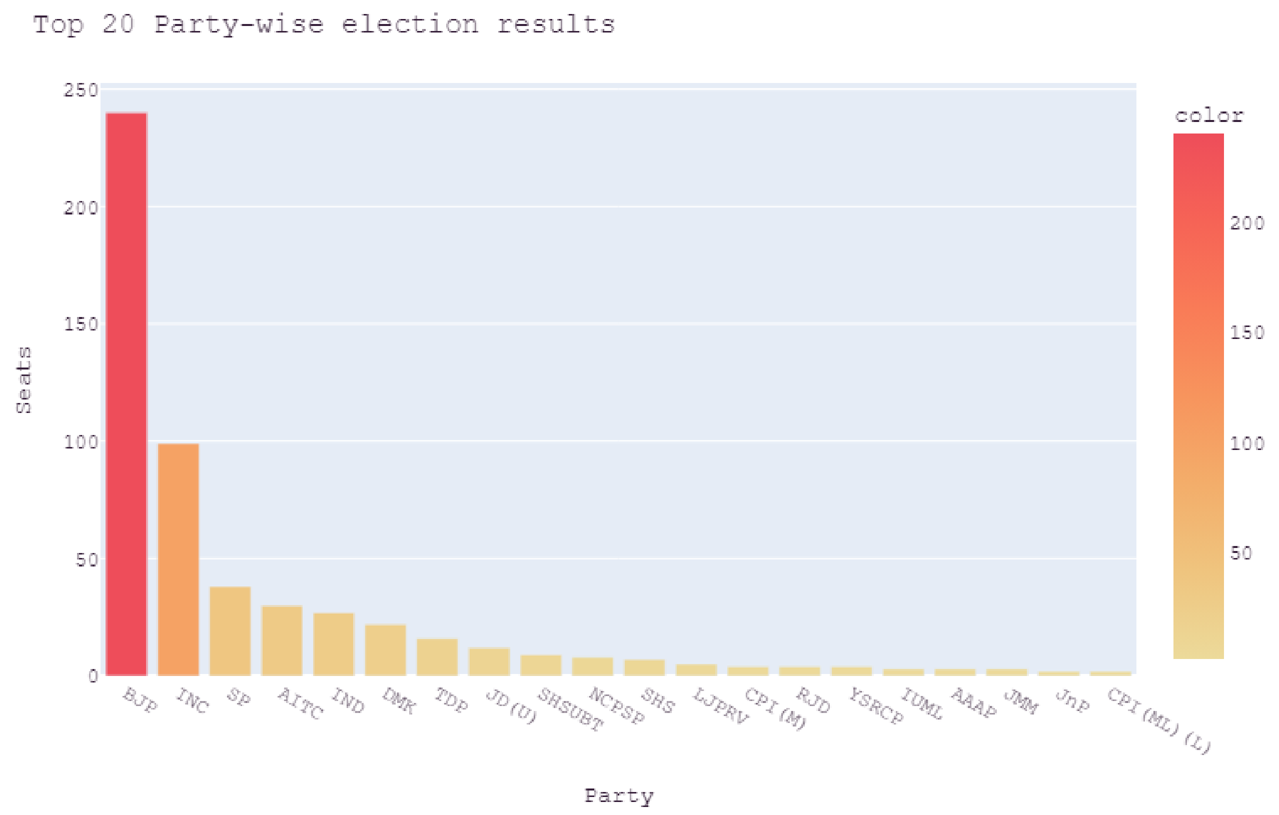


Figure 6.1: Top 20 Party-wise election results

- **Age Distribution:** A histogram depicted the distribution of candidates' ages, showing the prevalence of younger versus older candidates.

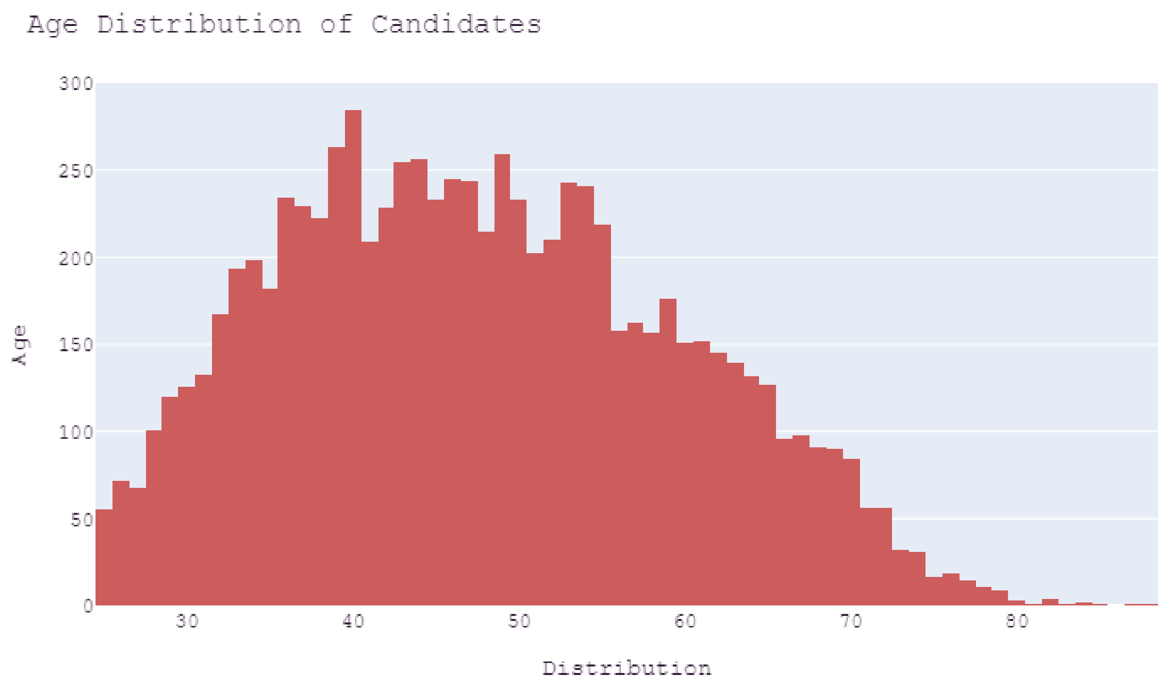


Figure 6.2: Age Distribution of Candidates

- **Caste Representation:** A bar chart illustrated the distribution of candidates across different caste categories, highlighting disparities in representation.

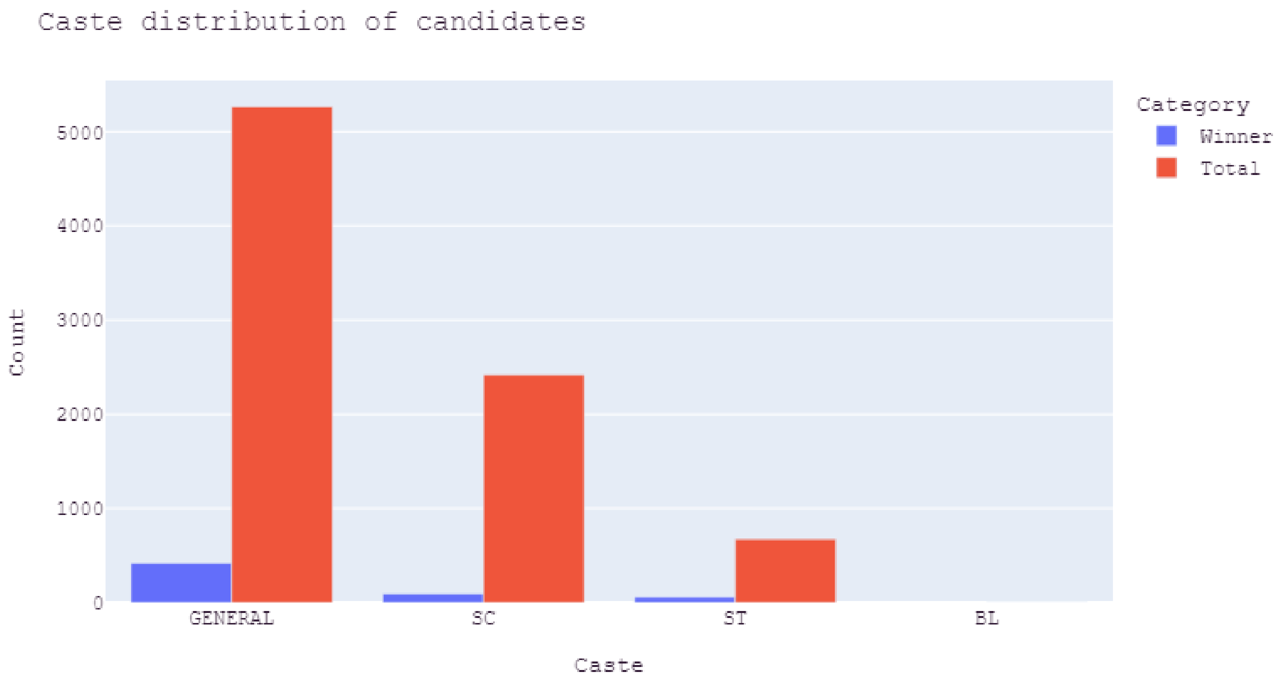


Figure 6.3: Caste distribution of candidates

- **Criminal Records Analysis:** Boxplots revealed how criminal records were distributed among different parties and their impact on election results.

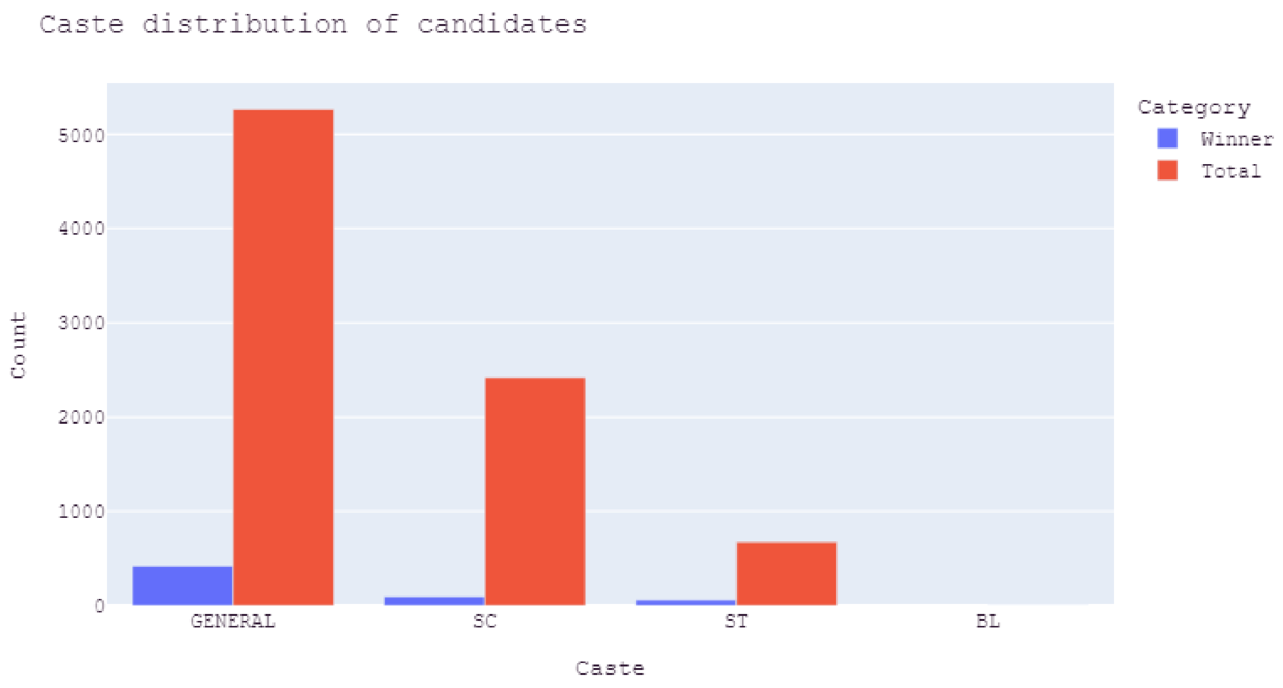


Figure 6.4: Top 10 Party-wise criminal records

The findings from EDA helped in selecting relevant features and understanding data patterns, leading to more informed model training.

## 6.6 Feature Selection Module

Feature importance was evaluated using XGBoost's feature importance analysis. The most influential attributes included party affiliation, candidate age, total votes, total electors, assets, liabilities, education level, criminal cases, gender, and category.

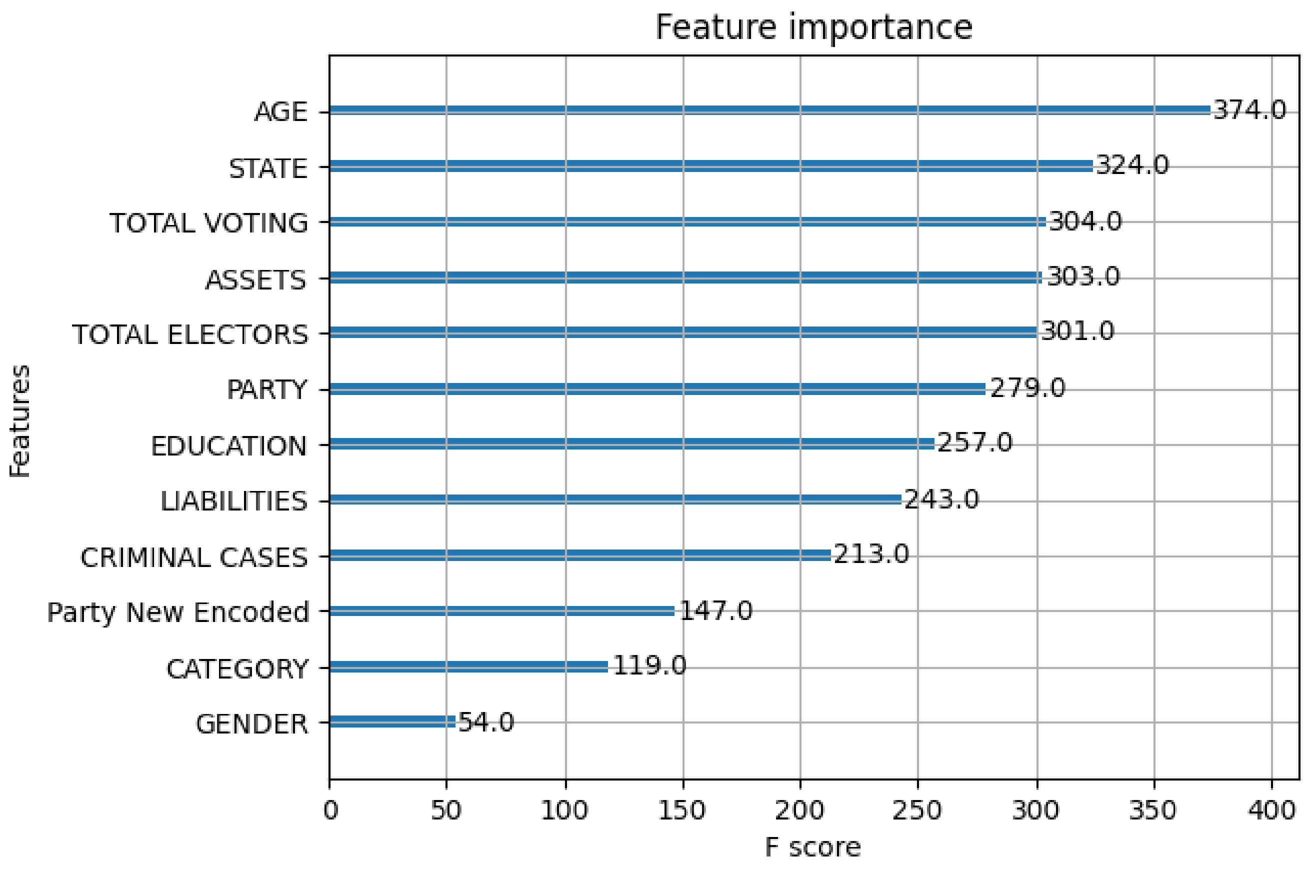


Figure 6.5: Feature Importance

## 6.7 Algorithm Details

This section describes the core algorithms used for election prediction, including their working principles, advantages and mathematical logic used in the project.

### 6.7.1 Algorithm 1: XGBoost (Extreme Gradient Boosting)

XGBoost is an advanced ensemble learning method based on decision trees. It is widely used for structured data problems due to its high accuracy, computational efficiency, and ability to handle complex datasets. In this project, XGBoost is applied for both feature selection and election prediction, as it effectively identifies key attributes influencing election outcomes.

### Working of XGBoost

XGBoost follows the gradient boosting framework, which iteratively refines weak learners (decision trees) to minimize prediction errors. The key steps include:

- **Boosting Mechanism** – Each new tree corrects errors made by the previous trees.
- **Weighted Error Correction** – Misclassified samples are assigned higher weights in subsequent iterations to reduce bias.
- **Parallelized Execution** – Unlike traditional boosting, XGBoost optimizes performance using parallel computing.
- **Regularization (L1 & L2)** – Prevents overfitting by penalizing overly complex models.

### Advantages of XGBoost in Election Prediction

- **Handles Missing Data** – Uses built-in mechanisms to deal with missing values.
- **Feature Importance Ranking** – Identifies key factors affecting election results.
- **Faster Execution** – Utilizes hardware acceleration and parallelization for efficiency.

### Mathematical Logic of XGBoost

**Objective Function** The objective function in XGBoost consists of two key components:

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{6.1}$$

where:

- $l(y_i, \hat{y}_i)$ is the loss function (e.g., log loss for classification), which measures the difference between actual and predicted values.
- $\Omega(f_k)$ is the regularization term that controls model complexity and prevents overfitting.

### Gradient Boosting with Second-Order Approximation

XGBoost optimizes predictions using Taylor expansion up to the second order:

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}, \quad h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2} \tag{6.2}$$

where:

- $g_i$ is the first-order gradient (similar to gradient descent).
- $h_i$ is the second-order Hessian (provides curvature information for better optimization).

Using these values, the model updates predictions iteratively:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + f_t(x_i) \tag{6.3}$$

where $f_t(x_i)$ is the newly added tree function that improves the previous prediction.

### Tree Structure and Leaf Splitting

XGBoost grows decision trees by splitting nodes in a way that maximizes information gain. The gain function is computed as:

$$\text{Gain} = \frac{1}{2}\left[\frac{(\sum g_i)^2}{\sum h_i + \lambda} - \frac{(\sum g_L)^2}{\sum h_L + \lambda} - \frac{(\sum g_R)^2}{\sum h_R + \lambda}\right] - \gamma \tag{6.4}$$

where:

- $\lambda$ is the regularization term, which controls model complexity.
- $\gamma$ is the pruning parameter, which prevents unnecessary splits.
- $\sum g_L$ and $\sum g_R$ represent the summation of gradients for the left and right child nodes, respectively.

If the gain is below a certain threshold, the split is not performed, preventing overfitting.

#### Final Prediction

Once all trees are trained, the final prediction is obtained by aggregating outputs from all trees:

$$\hat{Y} = \sum_{k=1}^{K} f_k(X) \tag{6.5}$$

Each tree $f_k(X)$ contributes to the final prediction, making XGBoost a strong ensemble learner.

### 6.7.2 Algorithm 2: Random Forest

Random Forest is a powerful ensemble learning algorithm based on decision trees. It is widely used for classification and regression tasks, making it an effective choice for election prediction. In this project, Random Forest is utilized to improve prediction accuracy and reduce overfitting by aggregating multiple decision trees.

#### Working of Random Forest

Random Forest operates by constructing multiple decision trees and aggregating their outputs. The key steps include:

- **Bootstrap Aggregation (Bagging)** – The model creates multiple subsets of the training data using random sampling with replacement.
- **Feature Randomness** – Instead of considering all features, each tree is trained on a random subset of features, improving generalization.
- **Tree Construction** – Each decision tree is independently trained using Gini impurity or entropy as the splitting criterion.
- **Voting Mechanism** – In classification tasks, the final prediction is determined using majority voting across all decision trees.

#### Advantages of Random Forest in Election Prediction

- **Reduces Overfitting** – Aggregation of multiple trees ensures robust predictions.
- **Handles Non-Linearity** – Effectively captures complex relationships in election data.

- **Feature Importance Ranking** – Identifies the most influential attributes in predicting election outcomes.
- **Handles Missing Data** – Can impute missing values using proximity-based methods.

**Mathematical Logic of Random Forest**

**1. Decision Tree Splitting Criteria**

Each tree in the Random Forest is built using a splitting criterion such as:

**Gini Impurity**

The Gini Impurity measures the probability of misclassification if we randomly pick a label from the dataset:

$$Gini = 1 - \sum_{i=1}^{C} p_i^2 \tag{6.6}$$

where:

- $C$ is the number of classes.
- $p_i$ is the probability of a sample belonging to class $i$.

Lower Gini values indicate purer nodes.

**Entropy (Information Gain)**

Entropy quantifies the uncertainty in the dataset:

$$Entropy = -\sum_{i=1}^{C} p_i \log_2 p_i \tag{6.7}$$

The tree selects the feature that maximizes Information Gain (IG):

$$IG = Entropy_{parent} - \sum_{k} \frac{|D_k|}{|D|} \times Entropy_k \tag{6.8}$$

where:

- $D$ is the dataset before the split.
- $D_k$ represents the subsets after the split.

**2. Bootstrap Sampling (Bagging Technique)**

Instead of training on the full dataset, Random Forest trains each tree on a randomly sampled subset:

$$D_{\text{bootstrap}} = \{X_1, X_2, ..., X_B\} \tag{6.9}$$

where $B$ is the number of bootstrapped datasets. This ensures that each tree gets a different perspective of the data, increasing diversity and reducing variance.

**3. Voting Mechanism for Final Prediction**

For classification tasks, Random Forest performs majority voting:

$$\hat{Y} = mode(Y_1, Y_2, ..., Y_T) \tag{6.10}$$

where:

- $Y_t$ is the prediction from the $t^{th}$ decision tree.

The most frequent predicted class is selected as the final output.

For regression tasks, it takes the average of predictions:

$$\hat{Y} = \frac{1}{T}\sum_{t=1}^{T} Y_t \tag{6.11}$$

where $T$ is the total number of trees.

**4. Feature Importance Calculation**

Random Forest ranks feature importance based on how much they contribute to reducing impurity across all trees. The importance score for a feature $f$ is computed as:

$$Importance(f) = \frac{1}{T}\sum_{t=1}^{T}(Gini_{\text{before}} - Gini_{\text{after}}) \tag{6.12}$$

A higher value indicates a more influential feature in the election prediction model.

**Final Prediction in Election Forecasting**

After training multiple decision trees and aggregating their outputs, the final election prediction is determined as follows:

$$\hat{Y} = \text{Majority Vote}\{f_1(X), f_2(X), ..., f_T(X)\} \tag{6.13}$$

where each function $f_t(X)$ represents an individual decision tree in the Random Forest.

### 6.7.3 Algorithm 3: Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. It is particularly effective in high-dimensional spaces and for problems where the classes are not linearly separable. In this project, SVM is used for election prediction by identifying a decision boundary that best separates winning and losing candidates based on various attributes.

**Working of Support Vector Machine (SVM)**

- **Hyperplane Selection** – SVM finds the optimal hyperplane that maximizes the margin between two classes (e.g., winning vs. losing candidates).
- **Support Vectors** – The data points that lie closest to the hyperplane are called support vectors, and they determine the position of the decision boundary.
- **Kernel Trick** – When the data is not linearly separable, SVM uses kernel functions to transform the data into a higher-dimensional space where it becomes separable.
- **Optimization with Lagrange Multipliers** – The decision boundary is determined by solving an optimization problem using quadratic programming.

**Advantages of SVM in Election Prediction**

- Effective in high-dimensional spaces – Can handle election data with many attributes.
- Robust to outliers – Uses support vectors to minimize classification errors.
- Works well for small to medium datasets – Suitable when election datasets are limited in size.
- Can model non-linearity using kernels – Useful if election outcomes depend on complex, non-linear factors.

### Mathematical Logic of Support Vector Machine (SVM)

**Defining the Hyperplane** For a binary classification problem, the decision boundary (hyperplane) is given by:

$$w^T x + b = 0 \tag{6.14}$$

where:

- $w$ is the weight vector (coefficients determining the hyperplane direction).
- $x$ is the input feature vector.
- $b$ is the bias term (intercepts the hyperplane).

For classification, we predict the class $y$ as:

$$y = \text{sign}(w^T x + b) \tag{6.15}$$

**Maximizing the Margin** SVM finds the hyperplane that maximizes the margin $M$ between two classes:

$$M = \frac{2}{\|w\|} \tag{6.16}$$

The margin is maximized by solving the following optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \tag{6.17}$$

Subject to the constraints:

$$y_i(w^T x_i + b) \geq 1, \quad \forall i \tag{6.18}$$

**Handling Non-Linearity: Kernel Trick** If the data is not linearly separable, SVM transforms it into a higher-dimensional space using a kernel function $K(x_i, x_j)$. Common kernel functions include:

- **Linear Kernel**:

$$K(x_i, x_j) = x_i^T x_j \tag{6.19}$$

- **Polynomial Kernel**:
$$K(x_i, x_j) = (x_i^T x_j + c)^d \tag{6.20}$$

- **Radial Basis Function (RBF) Kernel**:
$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{6.21}$$

- **Sigmoid Kernel**:
$$K(x_i, x_j) = \tanh(\alpha x_i^T x_j + c) \tag{6.22}$$

**Soft Margin SVM (Handling Misclassifications)** In real-world election data, perfect separation may not be possible. A soft margin SVM allows some misclassified points by introducing slack variables $\xi_i$:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \tag{6.23}$$

Subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{6.24}$$

where:

- $C$ is a regularization parameter controlling the trade-off between margin size and misclassification.
- Larger $C$ means fewer misclassifications but may lead to overfitting.

**Lagrange Dual Formulation (Optimization using Support Vectors)** SVM optimization is performed using Lagrange multipliers:

$$L(w, b, \alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{6.25}$$

where:

- $\alpha_i$ are Lagrange multipliers that define support vectors.
- Only support vectors (data points on the margin) contribute to the final decision boundary.

The final prediction is given by:

$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right) \tag{6.26}$$

**Final Prediction in Election Forecasting**

For a candidate's election prediction, SVM determines:

$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right) \tag{6.27}$$

- If $y = +1$, the candidate is predicted to win.
- If $y = -1$, the candidate is predicted to lose.

### 6.7.4 Algorithm 4: K-Nearest Neighbors (K-NN)

K-Nearest Neighbors (K-NN) is a non-parametric, instance-based learning algorithm used for both classification and regression. It is one of the simplest machine learning algorithms, relying on distance-based similarity to make predictions. In this project, K-NN is employed for election prediction, determining the likelihood of a candidate winning based on the historical data of similar candidates.

**Working of K-Nearest Neighbors (K-NN)**

- **Storing the Dataset** – K-NN does not create a model but memorizes all training instances.
- **Finding Neighbors** – Given a new test sample, K-NN computes its distance from all training points.
- **Selecting the K Nearest Neighbors** – The algorithm identifies the K closest data points.
- **Voting for the Majority Class** – For classification, K-NN assigns the test sample to the majority class among its K nearest neighbors.
- **Final Prediction** – The predicted label is the class with the highest occurrence among the selected neighbors.

**Advantages of K-NN in Election Prediction**

- Simple and easy to implement – No training phase, only distance computations.
- Effective for small datasets – Can work well for constituency-based analysis.
- No assumption about data distribution – Works for complex election data patterns.
- Adaptable to multi-class classification – Useful for predicting results in a multi-party election system.

**Mathematical Logic of K-NN**

**1. Distance Calculation**

To determine the nearest neighbors, K-NN uses a distance metric between two points $x_i$ and $x_j$. Commonly used distance functions include:

- **Euclidean Distance (Most Commonly Used in K-NN)**

  $$d(x_i, x_j) = \sqrt{\sum_{m=1}^{M} (x_{im} - x_{jm})^2}$$

  where $x_i$ and $x_j$ are feature vectors, and $M$ is the number of features.

- **Manhattan Distance**

  $$d(x_i, x_j) = \sum_{m=1}^{M} |x_{im} - x_{jm}|$$

  This is useful when attributes represent categories.

- **Minkowski Distance (Generalization of Euclidean and Manhattan Distance)**

  $$d(x_i, x_j) = \left( \sum_{m=1}^{M} |x_{im} - x_{jm}|^p \right)^{\frac{1}{p}}$$

  where:

  - $p = 1 \rightarrow$ Manhattan Distance
  - $p = 2 \rightarrow$ Euclidean Distance

For election prediction, Euclidean distance is the most commonly used metric.

**2. Selecting the Value of K**

The number of neighbors $K$ is a crucial hyperparameter:

- Small $K$ (e.g., $K = 1$ or $K = 3$) – More sensitive to noise but captures local structure well.
- Large $K$ (e.g., $K = 10$ or $K = 15$) – More stable but may smooth out class boundaries too much.

A common approach is to select $K$ using cross-validation to find the best value.

**3. Voting Mechanism for Classification**

Once K nearest neighbors are identified, the algorithm assigns a class label using majority voting:

$$\hat{y} = \arg\max_{c} \sum_{i \in N} 1(y_i = c)$$

where:

- $\hat{y}$ is the predicted class.
- $N$ is the set of $K$ nearest neighbors.
- $1(y_i = c)$ is an indicator function (1 if the neighbor belongs to class $c$, otherwise 0).

The class with the maximum votes is assigned to the test sample.

**4. Weighted Voting (Handling Class Imbalance)**

To improve predictions, K-NN can use distance-based weighting instead of simple majority voting. This ensures closer neighbors contribute more to the decision:

$$W_c = \sum_{i \in N} \frac{1(y_i = c)}{d(x, x_i)}$$

where:

- $W_c$ is the weighted vote for class $c$.
- $d(x, x_i)$ is the distance of neighbor $x_i$ from the test sample $x$.

This approach helps when some classes are underrepresented in the dataset (e.g., independent candidates in elections).

**5. Computational Complexity of K-NN**

The time complexity of K-NN is:

$$O(n \cdot M)$$

where:

- $n$ is the number of training samples.
- $M$ is the number of features.

For large election datasets, K-NN can become slow, but optimizations like KD-Trees and Ball Trees can improve efficiency.

**Final Prediction in Election Forecasting**

For a candidate's election prediction, K-NN finds similar candidates from historical data and assigns a win or lose label based on the majority of their closest neighbors.

- If most of the neighbors were winning candidates, the model predicts a **win** for the new candidate.
- If most of the neighbors were losing candidates, the model predicts a **loss**.

### 6.7.5 Algorithm 5: Logistic Regression

Logistic Regression is a statistical and machine learning algorithm used for binary classification problems. In election prediction, it is used to determine the probability of a candidate winning or losing based on various factors such as age, party affiliation, assets, criminal records, and constituency demographics.

**Working of Logistic Regression**

- **Linear Combination of Features** – The input features are combined into a weighted sum plus a bias term.
- **Logistic (Sigmoid) Function** – The weighted sum is passed through a sigmoid function to squash the output between 0 and 1.

- **Threshold-Based Classification** – If the probability is greater than 0.5, the candidate is predicted as a winner; otherwise, a loser.
- **Model Training** – The algorithm optimizes the weights using Maximum Likelihood Estimation (MLE).

**Advantages of Logistic Regression in Election Prediction**

- **Interpretable Model** – Provides clear insights into how different factors influence election outcomes.
- **Handles Linearly Separable Data Well** – If election outcomes are influenced by linearly separable factors, Logistic Regression works effectively.
- **Probability-Based Predictions** – Outputs probability scores instead of just class labels.
- **Efficient for Large Datasets** – Computationally less expensive compared to SVM or K-NN.

**Mathematical Logic of Logistic Regression**

**Hypothesis Function** Unlike Linear Regression, which predicts a continuous value, Logistic Regression models a probability using the sigmoid function:

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n)}} \tag{6.28}$$

where:

- $h_\theta(x)$ is the predicted probability that the candidate wins.
- $\theta_0$ is the bias term.
- $\theta_1, \theta_2, ..., \theta_n$ are the model coefficients (weights).
- $x_1, x_2, ..., x_n$ are the input features (candidate attributes).

This function ensures that outputs are bounded between 0 and 1, making it ideal for probability estimation.

**Decision Rule for Classification** The final prediction is based on a threshold:

$$\hat{y} = \begin{cases} 1, & \text{if } h_\theta(x) \geq 0.5 \text{ (Candidate wins)} \\ 0, & \text{if } h_\theta(x) < 0.5 \text{ (Candidate loses)} \end{cases}$$

The default threshold is 0.5, but it can be adjusted based on domain-specific requirements.

**Cost Function (Log Loss)** Logistic Regression uses the log loss function (binary cross-entropy):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[ y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i)) \right] \tag{6.29}$$

where:

- $m$ is the number of training samples.
- $y_i$ is the actual outcome (1 for win, 0 for loss).
- $h_\theta(x_i)$ is the predicted probability of winning.

This function penalizes incorrect predictions more heavily when the confidence is high, leading to a more robust model.

**Gradient Descent for Optimization** To find the optimal weights $\theta$, Gradient Descent is used to minimize the cost function:

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{ij} \tag{6.30}$$

where:

- $\alpha$ is the learning rate.
- $h_\theta(x_i) - y_i$ is the prediction error.

Variants like Stochastic Gradient Descent (SGD) and Batch Gradient Descent can be used based on dataset size.

**Regularization (Preventing Overfitting)** To avoid overfitting, L1 (Lasso) and L2 (Ridge) regularization are added:

**L2 Regularization (Ridge Regression)**

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\theta(x_i)) + (1-y_i)\log(1-h_\theta(x_i))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{6.31}$$

**L1 Regularization (Lasso Regression)**

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\theta(x_i)) + (1-y_i)\log(1-h_\theta(x_i))] + \frac{\lambda}{m}\sum_{j=1}^{n}|\theta_j| \tag{6.32}$$

**Multi-Class Logistic Regression (Softmax Regression)** For multi-class classification, the Softmax function generalizes Logistic Regression:

$$P(y=c|x) = \frac{e^{\theta_c^T x}}{\sum_{k=1}^{K} e^{\theta_k^T x}} \tag{6.33}$$

where:

- $K$ is the total number of classes (political parties).
- $P(y=c|x)$ is the probability of class $c$.

The class with the highest probability is assigned as the final prediction.

**Performance Metrics for Logistic Regression** Since Logistic Regression outputs probabilities, evaluation is done using:

- **Accuracy**: $\frac{\text{Correct Predictions}}{\text{Total Predictions}}$
- **Precision**: Measures how many predicted wins were actual wins.
- **Recall**: Measures how many actual wins were correctly predicted.
- **F1-Score**: Harmonic mean of precision and recall.
- **ROC-AUC Curve**: Measures how well the model separates winners and losers.

#### Final Prediction in Election Forecasting

For a given candidate, Logistic Regression predicts a win probability. If the probability $P(Win)$ is greater than 0.5, the candidate is classified as a winner, otherwise a loser.

**Example Prediction:**

- Candidate A ($P(Win) = 0.78$) $\Rightarrow$ Predicted Win
- Candidate B ($P(Win) = 0.34$) $\Rightarrow$ Predicted Loss

By adjusting the probability threshold, we can fine-tune model sensitivity to election outcomes.

### 6.7.6 Algorithm 6: CatBoost (Categorical Boosting)

CatBoost (Categorical Boosting) is a gradient boosting algorithm optimized for handling categorical features efficiently. It is particularly useful in election prediction, where party affiliation, constituency, and other categorical attributes significantly influence the outcome.

#### Working of CatBoost

- **Boosting Framework** – CatBoost is based on gradient boosting, where multiple weak learners (decision trees) are combined sequentially to improve prediction accuracy.
- **Categorical Feature Encoding** – Unlike traditional algorithms, CatBoost natively supports categorical data, reducing the need for extensive preprocessing.
- **Ordered Boosting** – Prevents target leakage by ensuring that each data point is learned only from past observations.
- **Oblivious Decision Trees** – Uses balanced trees where each level splits on the same feature, leading to fast inference and reduced overfitting.
- **L2 Regularization and Feature Importance** – Automatically ranks features by importance and applies regularization to avoid overfitting.

#### Advantages of CatBoost in Election Prediction

- Handles Categorical Data Natively – No need for one-hot encoding, making it efficient for political datasets.

- High Accuracy – Outperforms traditional boosting algorithms (XGBoost, LightGBM) on many structured datasets.
- Robust to Overfitting – Uses ordered boosting and regularization to maintain generalization.
- Fast Training and Inference – Optimized tree structure allows efficient training on large datasets.

**Mathematical Logic of CatBoost**

**Objective Function** CatBoost minimizes a loss function (Log Loss for classification, RMSE for regression) along with a regularization term:

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \sum_{j=1}^{m} \theta_j^2$$

where:

- $l(y_i, \hat{y}_i)$ is the loss function (cross-entropy for binary classification).
- $\lambda \sum_{j=1}^{m} \theta_j^2$ is the L2 regularization term to prevent overfitting.

**Ordered Boosting (Prevents Target Leakage)** Unlike standard gradient boosting, which uses all past and future observations, CatBoost orders the data and uses only past observations for each sample to avoid target leakage.

$$E[f(x_i)] = \frac{\sum_{j=1}^{i-1} f(x_j)}{i-1}$$

This ensures that model training does not leak future knowledge, making the predictions more robust.

**Categorical Feature Encoding (Efficient Handling of Categorical Data)** Instead of one-hot encoding, CatBoost uses a combination of:

**(a) Target Encoding**

For categorical variable $x_i$, the encoded value is:

$$E[f(x_i)] = \frac{\sum_{j=1}^{i-1} y_j}{i-1+\alpha}$$

where:

- $y_j$ are the target values for past observations.
- $\alpha$ is a smoothing parameter to avoid overfitting.

**(b) Frequency-Based Encoding**

$$\text{Encoded Value} = \frac{\text{Count of category in past data}}{\text{Total past observations}}$$

This encoding improves performance on categorical-heavy datasets like election results, where party affiliation, candidate names, and constituency names play a critical role.

**Oblivious Decision Trees (Balanced Tree Structure)** Unlike standard decision trees, CatBoost uses oblivious trees, where each level of the tree splits on the same feature for all nodes.

$$\text{Split Condition: } x_j \geq c$$

where:

- $x_j$ is the selected feature.
- $c$ is the optimal split threshold.

This structure improves parallel computation and reduces overfitting by maintaining a balanced tree.

**Gradient Boosting with Newton Step Approximation** Similar to XGBoost, CatBoost improves predictions by minimizing the second-order Taylor expansion of the loss function:

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}, \quad h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$$

The update rule for new predictions:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} - \eta \cdot \frac{g_i}{h_i}$$

where:

- $\eta$ is the learning rate.
- $g_i$ is the gradient (first derivative).

- $h_i$ is the Hessian (second derivative).

This ensures faster convergence and better optimization compared to standard gradient boosting methods.

**Feature Importance & Selection** CatBoost provides a feature importance ranking to determine the most influential variables in election outcomes. The importance score is calculated as:

$$\text{Feature Importance} = \sum_{t=1}^{T} \left| \frac{\partial L}{\partial x_t} \right|$$

where:

- $L$ is the loss function.
- $x_t$ is the feature.

This allows us to identify which attributes (e.g., party affiliation, assets, criminal cases, and voter turnout) have the highest impact on election results.

**Performance Metrics for CatBoost** Since CatBoost outputs probabilities, we evaluate its performance using:

- Log Loss (Binary Cross-Entropy) – Measures how well the predicted probabilities match actual outcomes.
- ROC-AUC Score – Evaluates the model's ability to distinguish between winners and losers.
- Accuracy, Precision, Recall, F1-score – Standard classification metrics.

**Final Prediction in Election Forecasting**

For a given candidate, CatBoost predicts a win probability. If the probability $P(\text{Win})$ is greater than 0.5, the candidate is classified as a winner; otherwise, a loser.

**Example Predictions:**

- Candidate A ($P(\text{Win}) = 0.82$) $\rightarrow$ Predicted Win
- Candidate B ($P(\text{Win}) = 0.45$) $\rightarrow$ Predicted Loss

The model's probability scores allow for rank-based forecasting, improving election analysis accuracy.

### 6.7.7 Algorithm 7: Gradient Boosting (GBM - Gradient Boosting Machine)

Gradient Boosting is a supervised machine learning algorithm that builds a strong predictive model by sequentially combining multiple weak learners (decision trees). It is widely used for election prediction due to its ability to capture complex patterns in the data.

**Working of Gradient Boosting**

- **Boosting Framework** – Unlike Random Forest (which builds trees in parallel), Gradient Boosting builds trees sequentially, where each tree corrects errors made by previous trees.
- **Gradient Descent Optimization** – Instead of simply weighting misclassified samples (as in AdaBoost), GBM minimizes a loss function using gradient descent to iteratively improve predictions.
- **Weighted Residual Learning** – Each new tree is trained to predict the residual errors of previous trees, gradually reducing prediction errors.
- **Regularization Techniques** – Prevents overfitting using learning rate, L1/L2 regularization, and tree pruning.

**Advantages of Gradient Boosting in Election Prediction**

- High Predictive Accuracy – Outperforms traditional machine learning models.
- Captures Complex Patterns – Effectively models non-linear relationships in electoral data.
- Feature Importance Ranking – Helps identify the most influential factors in predicting election outcomes.
- Handles Missing Data – Can work with incomplete datasets, making it robust for real-world applications.

### Mathematical Logic of Gradient Boosting

**Objective Function** Gradient Boosting minimizes a loss function $L(y, \hat{y})$ using an additive model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Where:

- $F_m(x)$ is the prediction at iteration $m$.
- $F_{m-1}(x)$ is the prediction from the previous step.
- $h_m(x)$ is the new weak learner (decision tree).
- $\gamma_m$ is the learning rate controlling the contribution of each tree.

**Gradient Descent Optimization** Each new tree is trained to minimize the loss function's gradient. The residual errors (negative gradients) at iteration $m$ are:

$$r_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$$

where:

- $r_{im}$ is the residual for sample $i$.
- $L(y, F)$ is the loss function (e.g., log loss for classification).
- $F_{m-1}(x_i)$ is the previous iteration's prediction.

The next tree $h_m(x)$ is trained to predict $r_{im}$, and we update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

**Decision Tree Construction in GBM** Each weak learner is a shallow decision tree, trained on the residuals. The tree splits are chosen to minimize:

$$\sum_{\text{leaves}} L(y, \hat{y})$$

where $L(y, \hat{y})$ is the loss function for classification or regression.

**Weight Optimization for New Trees** For each new tree, the optimal weight $\gamma_m$ is determined using line search:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

This ensures that each new tree minimally disturbs previous predictions while improving accuracy.

**Regularization to Prevent Overfitting** Gradient Boosting is prone to overfitting, so regularization techniques are applied:

- **Learning Rate (Shrinkage)** – Controls the contribution of each tree:

  $$F_m(x) = F_{m-1}(x) + \eta h_m(x)$$

  where $\eta$ (learning rate) is a small value (e.g., 0.1) to prevent large updates.

- **Tree Pruning (Max Depth)** – Limits tree depth to prevent overfitting. Typically, GBM trees have a depth of 3-5 levels.

- **L1 & L2 Regularization** – Adds a penalty to large weights:

  $$L_{\text{regularized}} = L + \lambda \sum |w_j| + \alpha \sum w_j^2$$

  where:

  - $\lambda$ controls L1 regularization (sparsity).
  - $\alpha$ controls L2 regularization (smoothness).

**Feature Importance in GBM**

GBM provides a ranking of feature importance by measuring how much each feature contributes to reducing loss. The importance score is calculated as:

$$\text{Feature Importance} = \sum_{\text{splits on feature } j} \Delta\text{Loss}$$

This helps in election analysis by identifying key factors such as:

- Party affiliation
- Total votes received
- Candidate's financial assets
- Criminal record impact

**Performance Metrics for Gradient Boosting**

Since GBM predicts probabilities, its performance is evaluated using:

- **Log Loss (Binary Cross-Entropy)** – Measures how well predicted probabilities match actual results.
- **ROC-AUC Score** – Determines the model's ability to distinguish between winners and losers.
- **Precision, Recall, F1-score** – Evaluates classification performance.

**Final Prediction in Election Forecasting**

For each candidate, GBM outputs a win probability $P(\text{Win})$. If $P(\text{Win}) > 0.5$, the candidate is classified as a winner, otherwise as a loser.

**Example Predictions:**

- Candidate A ($P(\text{Win}) = 0.78$) $\rightarrow$ Predicted Win
- Candidate B ($P(\text{Win}) = 0.35$) $\rightarrow$ Predicted Loss

The model can also rank candidates by probability, allowing better election analysis.

### 6.7.8 Algorithm 8: AdaBoost (Adaptive Boosting)

Adaptive Boosting (AdaBoost) is a sequential ensemble learning technique that combines multiple weak classifiers to create a strong classifier. It assigns different weights to misclassified samples, forcing the model to focus on hard-to-classify cases.

**Working of AdaBoost**

- **Initialize Equal Weights** – All training samples start with equal weight.
- **Train Weak Learners** – A weak classifier (usually a decision stump) is trained to minimize errors.
- **Calculate Error & Update Weights** – Samples that are misclassified receive higher weights, making them more influential in the next iteration.
- **Combine Weak Learners** – Final prediction is a weighted sum of all weak classifiers.

**Advantages of AdaBoost in Election Prediction**

- ✓Focuses on Misclassified Data – Improves accuracy on challenging election patterns.
- ✓Reduces Bias – Works well with noisy datasets.
- ✓Simple & Fast – Efficient for small datasets.
- ✓Works with Any Weak Learner – Often used with decision trees or logistic regression.

**Mathematical Logic of AdaBoost**

**Weight Initialization** Each training sample $i$ is given an initial weight:

$$w_i(1) = \frac{1}{N}$$

where $N$ is the total number of training samples.

**Train Weak Learners** A weak classifier $h_t(x)$ is trained at each iteration $t$ to minimize the weighted classification error:

$$\epsilon_t = \sum_{i=1}^{N} w_i(t) I(y_i \neq h_t(x_i))$$

where:

- $y_i$ is the true label.
- $h_t(x_i)$ is the predicted label.
- $I(y_i \neq h_t(x_i))$ is an indicator function that equals 1 if misclassified and 0 otherwise.

**Compute Classifier Weight $\alpha_t$** Each weak classifier is assigned a weight:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

where:

- $\alpha_t$ determines the importance of classifier $h_t(x)$.
- If $\epsilon_t$ is low (classifier is accurate), $\alpha_t$ is high, meaning this classifier is trusted more.

**Update Sample Weights** Misclassified samples receive higher weights to ensure they get more attention in the next iteration:

$$w_i(t+1) = w_i(t) \cdot \exp(\alpha_t I(y_i \neq h_t(x_i)))$$

After updating, weights are normalized so they sum to 1:

$$w_i(t+1) = \frac{w_i(t+1)}{\sum w_i(t+1)}$$

This ensures the total sample weight remains constant.

**Final Prediction** The final model combines all weak classifiers based on their importance:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

- If the sum is positive, predict 1 (win).
- If the sum is negative, predict 0 (loss).

Each classifier's influence is weighted by $\alpha_t$, so more accurate classifiers have higher impact on the final decision.

**Regularization in AdaBoost**

Since AdaBoost is prone to overfitting, regularization techniques are used:

- **Limiting the Number of Iterations** $T$ – Too many classifiers may fit noise.
- **Using Stumps (Depth-1 Trees)** – Prevents overly complex models.

- **Learning Rate** $\eta$ – Adjusts classifier impact:

$$\alpha'_t = \eta \cdot \alpha_t, \quad 0 < \eta < 1$$

where smaller $\eta$ prevents abrupt weight changes.

**Feature Importance in AdaBoost**

AdaBoost naturally ranks features by tracking how often a feature contributes to tree splits. The importance score is:

$$\text{Feature Importance} = \sum_{t=1}^{T} \alpha_t \cdot I(\text{Feature Used in } h_t)$$

This helps identify key factors like:

- Party Affiliation
- Vote Share
- Criminal Cases
- Candidate Assets

**Performance Metrics for AdaBoost**

Since AdaBoost outputs binary classifications, its performance is evaluated using:

- ✓Accuracy – Measures correct predictions.
- ✓Precision & Recall – Identifies class imbalance effects.
- ✓ROC-AUC Score – Assesses ability to distinguish winners from losers.

**Final Prediction in Election Forecasting**

For each candidate, AdaBoost assigns a win probability. If the final score is positive, the candidate is predicted as winner, else loser.

**Example:**

- Candidate A (Win Score = +3.2) $\rightarrow$ Predicted Win
- Candidate B (Win Score = -1.7) $\rightarrow$ Predicted Loss

This method ensures misclassified candidates receive extra focus, improving accuracy.

### 6.7.9 Algorithm 9: LightGBM (Light Gradient Boosting Machine)

LightGBM (Light Gradient Boosting Machine) is a fast and efficient gradient boosting algorithm that is optimized for large datasets. It is an improvement over traditional gradient boosting methods such as XGBoost, designed to handle high-dimensional data with lower memory usage and faster computation.

#### Working of LightGBM

- **Histogram-Based Learning** – Instead of scanning all data points, LightGBM groups feature values into discrete bins, making training much faster.
- **Leaf-Wise Growth Strategy** – Unlike traditional boosting methods that grow trees level-wise, LightGBM grows them leaf-wise, focusing on the most informative splits first.
- **Gradient-Based One-Side Sampling (GOSS)** – Instead of using all data points, LightGBM selects only important samples to calculate gradients, reducing computation.
- **Exclusive Feature Bundling (EFB)** – Combines sparse features to reduce dimensionality, improving efficiency.

#### Advantages of LightGBM in Election Prediction

- ✓Faster Training – Can process millions of election records efficiently.
- ✓Handles Large & Sparse Data – Works well with datasets containing many categorical and numerical features.
- ✓Better Accuracy – Leaf-wise growth helps in capturing complex patterns in election data.
- ✓Lower Memory Usage – Uses histogram-based learning, requiring less RAM.

#### Mathematical Logic of LightGBM

LightGBM is based on gradient boosting, where new trees are added iteratively to minimize prediction error.

**Objective Function** The objective function of LightGBM consists of two terms:

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{6.34}$$

where:

- $l(y_i, \hat{y}_i)$ is the loss function (e.g., log loss for classification).
- $\Omega(f_k)$ is the regularization term to prevent overfitting.

**Gradient Boosting Update Rule** At each iteration, a new tree is added to improve predictions using Taylor expansion:

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}, \quad h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2} \tag{6.35}$$

where:

- $g_i$ is the first-order gradient (how much to adjust predictions).
- $h_i$ is the second-order gradient (measures curvature for better optimization).

The new prediction update is:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + f_t(x_i) \tag{6.36}$$

where $f_t(x_i)$ is the newly added tree.

**Leaf-Wise Tree Growth Strategy** Unlike XGBoost, which grows level-wise (expanding all nodes at the same depth), LightGBM expands the most promising leaf first to maximize the information gain:

$$\text{Gain} = \frac{1}{2} \left[ \frac{\left(\sum g_i\right)^2}{\sum h_i + \lambda} - \frac{\left(\sum g_L\right)^2}{\sum h_L + \lambda} - \frac{\left(\sum g_R\right)^2}{\sum h_R + \lambda} \right] - \gamma \tag{6.37}$$

where:

- $g$ and $h$ are the gradients and Hessians.
- $\lambda$ is the regularization term.
- $\gamma$ controls pruning, preventing unnecessary splits.

**Gradient-Based One-Side Sampling (GOSS)** Instead of using all training samples, LightGBM selects only important samples based on gradient magnitude:

$$G_{\text{selected}} = \sum_{i \in \text{high-gradients}} g_i + \frac{1}{r} \sum_{i \in \text{low-gradients}} g_i \tag{6.38}$$

where $r$ is the sampling ratio.

**Exclusive Feature Bundling (EFB)** Election datasets often contain many categorical variables (e.g., state, constituency, party). Many of these features are sparse (i.e., contain many zeros).

LightGBM groups sparse features into fewer bundled features, reducing dimensionality:

$$\text{EFB Size} = \sum_{j=1}^{m} I(F_j \neq 0) \tag{6.39}$$

This makes training faster without losing information.

### Feature Importance in LightGBM

LightGBM provides feature importance scores based on how frequently a feature is used for splitting:

$$\text{Feature Importance} = \sum_{t=1}^{T} I(\text{Feature Used in } f_t) \tag{6.40}$$

In election prediction, important features might include:

- Total Votes Received
- Party Affiliation
- Candidate's Assets
- Criminal Cases

### Regularization in LightGBM

LightGBM avoids overfitting using:

- **L1/L2 Regularization:** Adds penalty terms $\lambda_1 |w| + \lambda_2 w^2$ to tree weights.

- **Early Stopping:** Stops training if performance does not improve after a set number of rounds.
- **Max Depth Limitation:** Prevents trees from growing too deep.

**Performance Metrics for LightGBM**

Since LightGBM is a classification model, we evaluate it using:

- ✓**Accuracy** – Percentage of correctly predicted winners.
- ✓**Precision & Recall** – To handle class imbalances.
- ✓**ROC-AUC Score** – Measures how well the model separates winners from losers.

**Final Prediction in Election Forecasting**

For each candidate, LightGBM assigns a probability score $p$, representing their likelihood of winning.

$$\hat{y}_i = \text{sigmoid}\left(\sum_{t=1}^{T} f_t(x_i)\right) \tag{6.41}$$

If $p > 0.5$, the candidate is predicted as a winner; otherwise, they are predicted as a loser.
**Example:**

- Candidate A (Win Probability = 0.78) → Predicted Win
- Candidate B (Win Probability = 0.42) → Predicted Loss

### 6.7.10 Algorithm 10: Decision Tree

A Decision Tree is a supervised learning algorithm used for both classification and regression. It mimics human decision-making by splitting data based on feature conditions, forming a tree-like structure. In the context of election prediction, Decision Trees help determine winning probabilities based on features like candidate's party, assets, criminal cases, and total votes.

### Working of Decision Tree

- **Splitting** – The dataset is divided into subsets based on feature conditions (e.g., ”Is the candidate from Party X?”).
- **Tree Construction** – The algorithm recursively splits the data until stopping criteria are met (e.g., max depth is reached).
- **Prediction** – A new candidate's feature values are compared against the tree structure to determine the predicted outcome.

### Advantages of Decision Trees in Election Prediction

- **Interpretable** – Provides a visual structure for analyzing election outcomes.
- **Handles Categorical & Numerical Data** – Works well with attributes like ”party” and ”total votes”.
- **Feature Selection** – Identifies the most important factors influencing election results.

### Mathematical Logic of Decision Trees

Decision Trees use entropy and information gain (or Gini impurity) to decide how to split nodes.

**Entropy (Measure of Disorder)** Entropy measures uncertainty in a dataset:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

where:

- $H(S)$ is the entropy of dataset $S$.
- $p_i$ is the probability of class $i$ (e.g., winner or loser).

A dataset with equal winners and losers has high entropy, while a dataset with only winners has low entropy.

**Example:**

- If all candidates win: $H = 0$ (pure).
- If 50% win and 50% lose: $H = 1$ (maximum uncertainty).

**Information Gain (Choosing the Best Split)** To decide the best feature for splitting, we calculate Information Gain:

$$IG(S, A) = H(S) - \sum_{v \in A} \frac{|S_v|}{|S|} H(S_v)$$

where:

- $IG(S, A)$ is the information gain after splitting on attribute $A$.
- $S_v$ is the subset of $S$ for which $A = v$.

A higher Information Gain means the feature better separates winners from losers, making it a good split.

**Example:** Splitting on party affiliation may yield high information gain if certain parties have higher winning chances.

**Gini Impurity (Alternative to Entropy)** Instead of entropy, Gini impurity can be used:

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

A low Gini index means a pure dataset. The split with the lowest Gini is chosen.

**Recursive Splitting & Stopping Criteria** The process of splitting continues recursively until one of the following conditions is met:

- **Maximum depth reached** – Prevents overfitting.
- **Minimum samples per leaf** – Ensures each leaf has enough data.
- **Entropy/Gini below threshold** – Stops when the subset is pure.

**Prediction in Election Forecasting**

Once the tree is built, predictions are made by traversing the tree based on a candidate's attributes.

**Example Decision Tree:**

```
              Party?
             /      \
          BJP        Congress
         /   \       /      \
      Assets?  Age?  Cases?  Votes?
```

If a candidate is from BJP, has high assets, and few criminal cases, they are predicted to win. If a candidate is from Congress, has low votes, and has criminal cases, they are predicted to lose.

Mathematically, the final decision is:

$$y_i = \arg\max_c p(c \mid X_i)$$

where $p(c \mid X_i)$ is the probability of class $c$ given feature values $X_i$.

**Overfitting & Regularization**

Decision Trees can overfit, meaning they memorize training data instead of generalizing. To prevent this:

- **Pruning** – Removes unnecessary branches using a cost-complexity function:

  $$C(T) = \sum_i H(S_i) + \alpha|T|$$

  where $\alpha$ controls pruning severity.

- **Max Depth Limitation** – Prevents trees from growing too deep.

- **Min Samples Split** – Ensures a node must have at least $n$ samples to split.

**Performance Metrics for Decision Trees**

Decision Trees are evaluated using:

- **Accuracy** – Measures how many winners were correctly classified.

- **Precision & Recall** – Identifies misclassified candidates.

- **ROC Curve** – Measures classification power.

## 6.8 Model Training and Evaluation Module

The Model Training and Evaluation Module is a crucial phase in the election prediction system. This stage involves training multiple machine learning models on historical election data and evaluating their performance based on various metrics to identify the most effective model.

### 6.8.1 Training Machine Learning Models

To ensure a robust and accurate election prediction system, ten different machine learning algorithms were evaluated:

- Random Forest
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Logistic Regression
- XGBoost
- CatBoost
- Gradient Boosting
- AdaBoost
- LightGBM
- Decision Tree

Each model was trained using a train-test split approach, where:

- 70% of the data was used for training the model.
- 30% of the data was used for testing and evaluating the model's performance.

Additionally, **10-Fold Cross-Validation** was applied to improve generalization and prevent overfitting.

### 6.8.2 Evaluation Metrics

After training, the models were assessed using four key evaluation metrics:

**1. Accuracy**

Measures the proportion of correctly classified election outcomes.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.42}$$

where:

- $TP$ (True Positives): Correctly predicted winning candidates.
- $TN$ (True Negatives): Correctly predicted losing candidates.
- $FP$ (False Positives): Incorrectly predicted winners.
- $FN$ (False Negatives): Incorrectly predicted losers.

**2. Precision**

Indicates how many predicted winners were actually winners.

$$Precision = \frac{TP}{TP + FP} \tag{6.43}$$

**3. Recall (Sensitivity)**

Measures how well the model identified actual winners.

$$Recall = \frac{TP}{TP + FN} \tag{6.44}$$

**4. F1-Score**

A balance between precision and recall, useful for handling imbalanced datasets.

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6.45}$$

After evaluating these metrics, the model with the highest accuracy and best overall performance was selected for making final predictions.

## 6.9 Model Selection and Prediction Module

The Model Selection and Prediction Module is responsible for choosing the best-performing model and using it to forecast election results.

### 6.9.1 Selection of the Best Model

Among all models tested, **CatBoost** was identified as the best-performing model due to:

- **High Accuracy**: Outperformed other models on validation data.
- **Efficient Handling of Categorical Data**: Essential for election prediction.
- **Better Generalization**: Robust to overfitting compared to other boosting models.

The final selected model was then used for predicting the election results based on candidate attributes such as age, criminal record, assets, education, and party affiliation.

### 6.9.2 Final Model Evaluation

After selecting the best model, further evaluation was conducted using:

**1. Confusion Matrix**

- Analyzes the model's classification performance.
- Helps in identifying errors in predictions, such as false positives and false negatives.

| | **Predicted Winner** | **Predicted Loser** |
|---|---|---|
| **Actual Winner** | True Positives (TP) | False Negatives (FN) |
| **Actual Loser** | False Positives (FP) | True Negatives (TN) |

Table 6.1: Confusion Matrix for Model Evaluation

**2. ROC Curve (Receiver Operating Characteristic Curve)**

- Measures the trade-off between sensitivity (TPR) and specificity (FPR).
- AUC-ROC Score was calculated to evaluate the model's classification power.
- AUC close to 1.0 indicates a highly accurate model.

**3. Classification Report**

Provides detailed performance insights, including:

- **Precision**
- **Recall**
- **F1-Score**

### 6.9.3 Final Prediction and Interpretation

Using the selected model, final election predictions were made.

**Final Outcome:**

- Winning probabilities were computed for each candidate.
- Top political parties were displayed along with their predicted chances of winning.
- Constituencies with close competition were identified.
- Key factors affecting election results were analyzed.

# Chapter 7

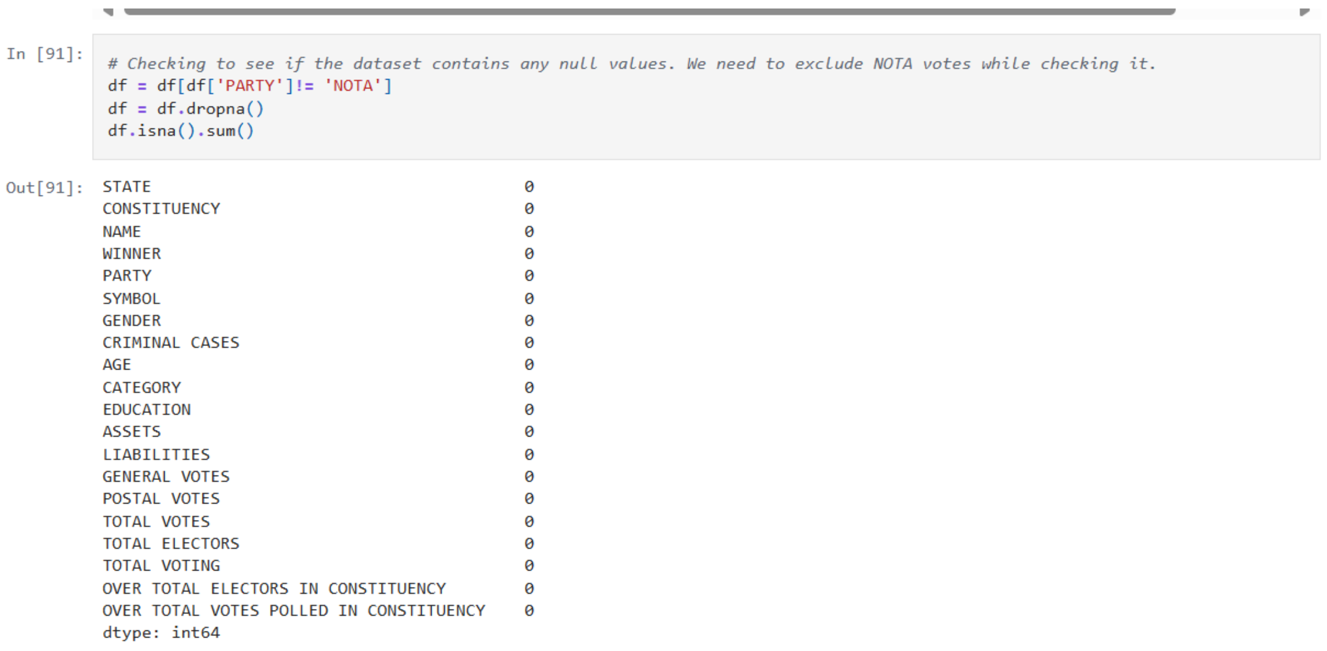
# Software Testing

## 7.1 Type of Testing

### 7.1.1 Model Performance Testing

- Measures how many predictions are correct.
- Suitable for balanced datasets but can be misleading for imbalanced ones.
- **Metric:** Accuracy = (Correct Predictions / Total Predictions) .
- **Precision:** Focuses on how many predicted winners are actually correct.
- **Recall:** Measures how well the model captures all actual winners.
- **F1-Score:** Harmonic mean of Precision & Recall, used for imbalanced datasets.
- Uses past election data from different timeframes/geographies to test generalization.
- Splits data into **training and testing** multiple times to evaluate consistency.
- Identifying anomalies in voting trends, turnout rates, or polling data.
- Ensuring that no **future election results** are accidentally used in training data.
- Tests the model on past elections to see if it would have predicted correctly.

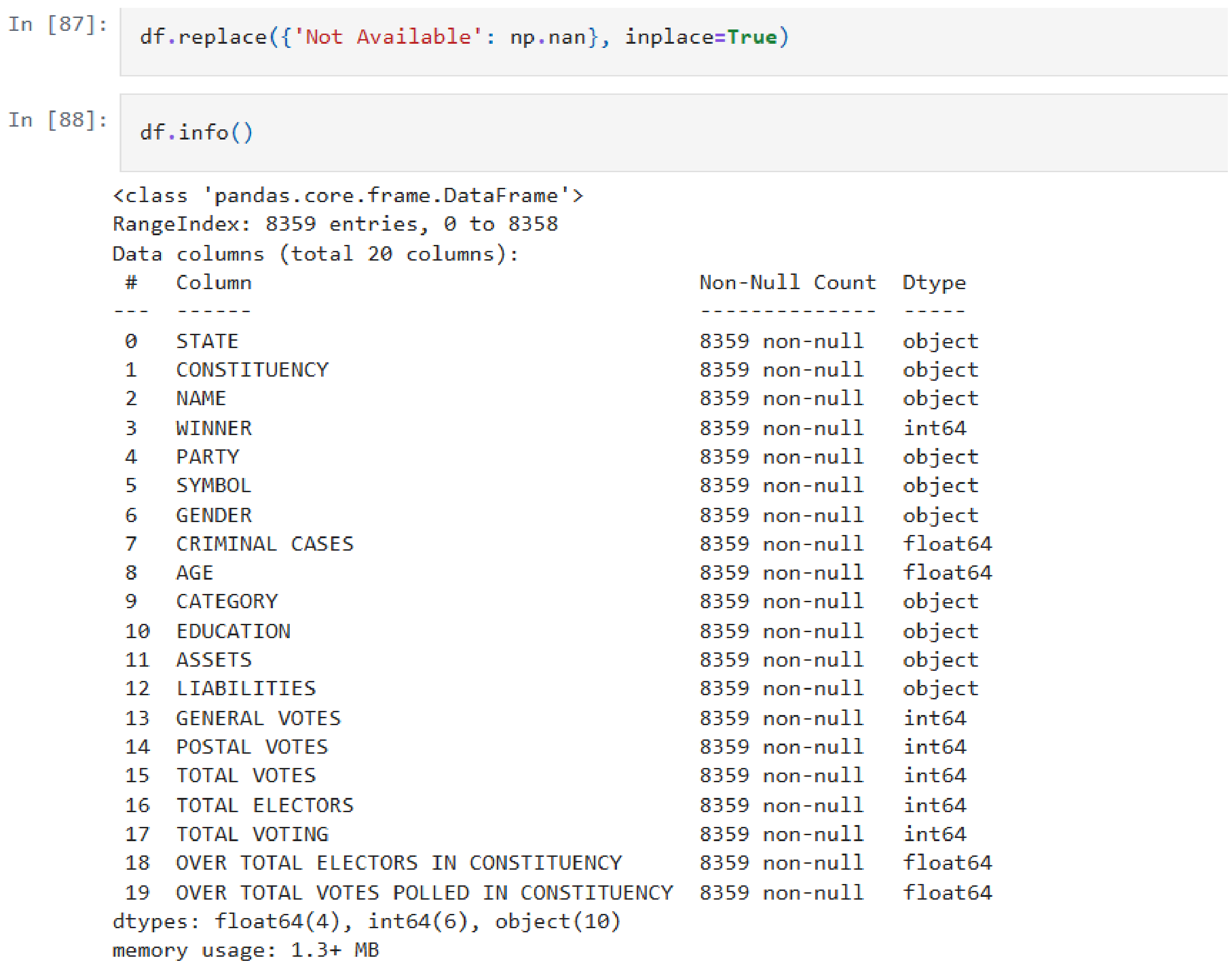
## 7.2 Test Results



```
In [91]: # Checking to see if the dataset contains any null values. We need to exclude NOTA votes while checking it.
         df = df[df['PARTY']!= 'NOTA']
         df = df.dropna()
         df.isna().sum()
```

```
Out[91]: STATE                                     0
         CONSTITUENCY                              0
         NAME                                      0
         WINNER                                    0
         PARTY                                     0
         SYMBOL                                    0
         GENDER                                    0
         CRIMINAL CASES                            0
         AGE                                       0
         CATEGORY                                  0
         EDUCATION                                 0
         ASSETS                                    0
         LIABILITIES                               0
         GENERAL VOTES                             0
         POSTAL VOTES                              0
         TOTAL VOTES                               0
         TOTAL ELECTORS                            0
         TOTAL VOTING                              0
         OVER TOTAL ELECTORS IN CONSTITUENCY       0
         OVER TOTAL VOTES POLLED IN CONSTITUENCY   0
         dtype: int64
```

Figure 7.1: TestCase1

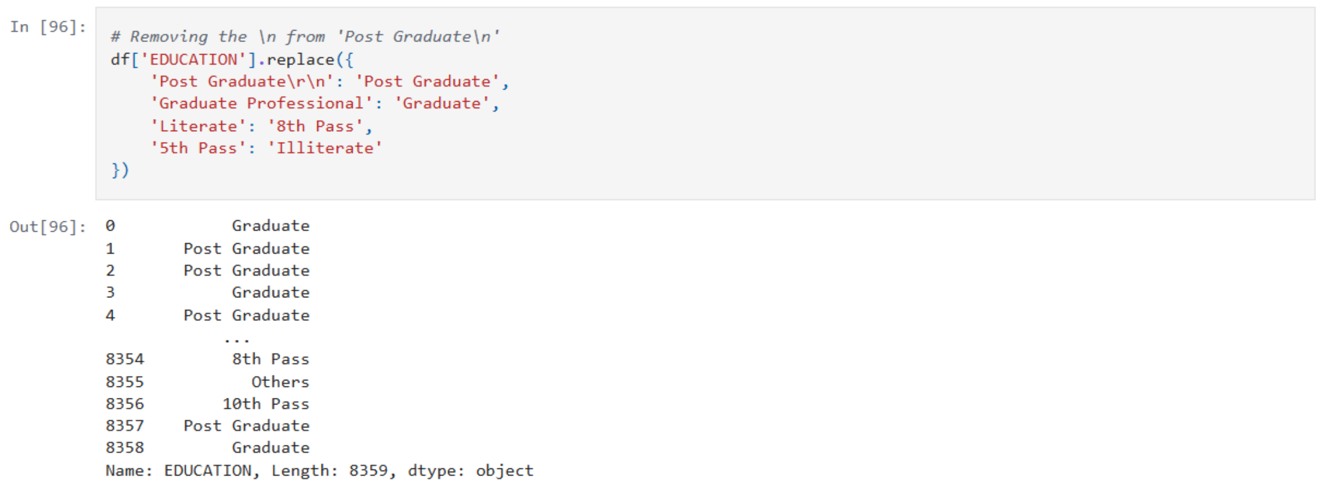


```
In [87]: df.replace({'Not Available': np.nan}, inplace=True)

In [88]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8359 entries, 0 to 8358
Data columns (total 20 columns):
 #   Column                                   Non-Null Count  Dtype  
---  ------                                   --------------  -----  
 0   STATE                                    8359 non-null   object 
 1   CONSTITUENCY                             8359 non-null   object 
 2   NAME                                     8359 non-null   object 
 3   WINNER                                   8359 non-null   int64  
 4   PARTY                                    8359 non-null   object 
 5   SYMBOL                                   8359 non-null   object 
 6   GENDER                                   8359 non-null   object 
 7   CRIMINAL CASES                           8359 non-null   float64
 8   AGE                                      8359 non-null   float64
 9   CATEGORY                                 8359 non-null   object 
 10  EDUCATION                                8359 non-null   object 
 11  ASSETS                                   8359 non-null   object 
 12  LIABILITIES                              8359 non-null   object 
 13  GENERAL VOTES                            8359 non-null   int64  
 14  POSTAL VOTES                             8359 non-null   int64  
 15  TOTAL VOTES                              8359 non-null   int64  
 16  TOTAL ELECTORS                           8359 non-null   int64  
 17  TOTAL VOTING                             8359 non-null   int64  
 18  OVER TOTAL ELECTORS IN CONSTITUENCY      8359 non-null   float64
 19  OVER TOTAL VOTES POLLED IN CONSTITUENCY  8359 non-null   float64
dtypes: float64(4), int64(6), object(10)
memory usage: 1.3+ MB
```

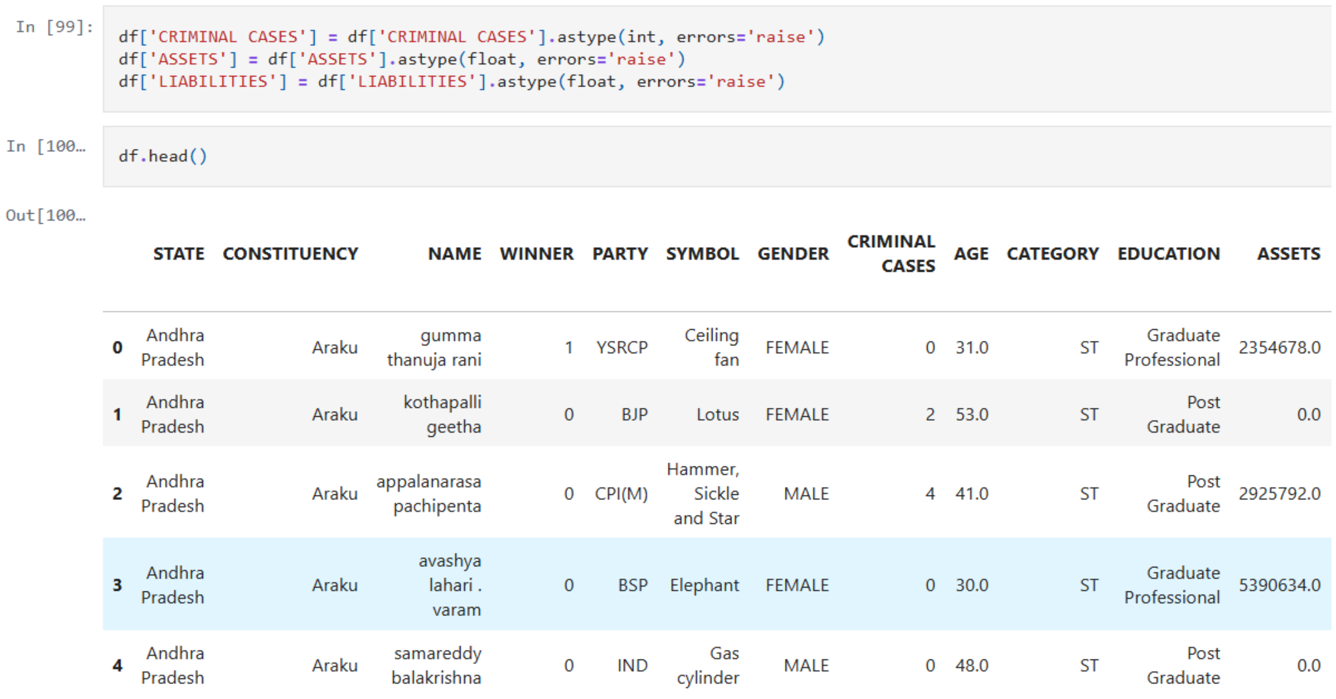
Figure 7.2: Testcase2



```
In [96]:  # Removing the \n from 'Post Graduate\n'
          df['EDUCATION'].replace({
              'Post Graduate\r\n': 'Post Graduate',
              'Graduate Professional': 'Graduate',
              'Literate': '8th Pass',
              '5th Pass': 'Illiterate'
          })

Out[96]:  0            Graduate
          1       Post Graduate
          2       Post Graduate
          3            Graduate
          4       Post Graduate
                      ...
          8354         8th Pass
          8355           Others
          8356        10th Pass
          8357    Post Graduate
          8358         Graduate
          Name: EDUCATION, Length: 8359, dtype: object
```

Figure 7.3: Testcase3



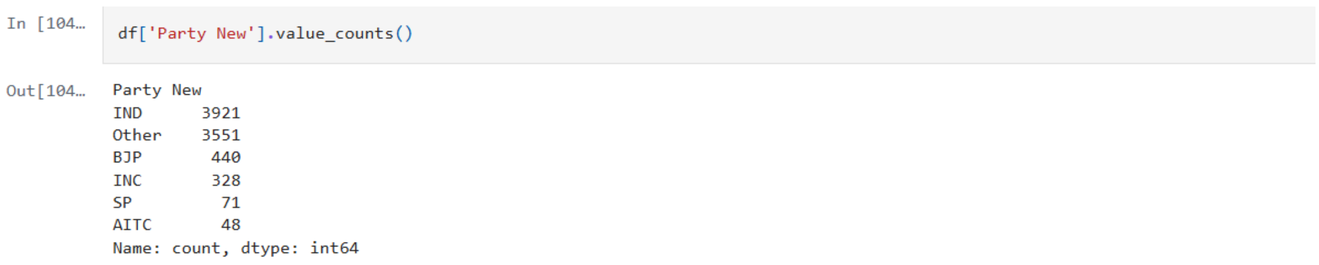
```
In [99]:  df['CRIMINAL CASES'] = df['CRIMINAL CASES'].astype(int, errors='raise')
          df['ASSETS'] = df['ASSETS'].astype(float, errors='raise')
          df['LIABILITIES'] = df['LIABILITIES'].astype(float, errors='raise')

In [100…  df.head()

Out[100…
```

| | STATE | CONSTITUENCY | NAME | WINNER | PARTY | SYMBOL | GENDER | CRIMINAL CASES | AGE | CATEGORY | EDUCATION | ASSETS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Andhra Pradesh | Araku | gumma thanuja rani | 1 | YSRCP | Ceiling fan | FEMALE | 0 | 31.0 | ST | Graduate Professional | 2354678.0 |
| **1** | Andhra Pradesh | Araku | kothapalli geetha | 0 | BJP | Lotus | FEMALE | 2 | 53.0 | ST | Post Graduate | 0.0 |
| **2** | Andhra Pradesh | Araku | appalanarasa pachipenta | 0 | CPI(M) | Hammer, Sickle and Star | MALE | 4 | 41.0 | ST | Post Graduate | 2925792.0 |
| **3** | Andhra Pradesh | Araku | avashya lahari . varam | 0 | BSP | Elephant | FEMALE | 0 | 30.0 | ST | Graduate Professional | 5390634.0 |
| **4** | Andhra Pradesh | Araku | samareddy balakrishna | 0 | IND | Gas cylinder | MALE | 0 | 48.0 | ST | Post Graduate | 0.0 |

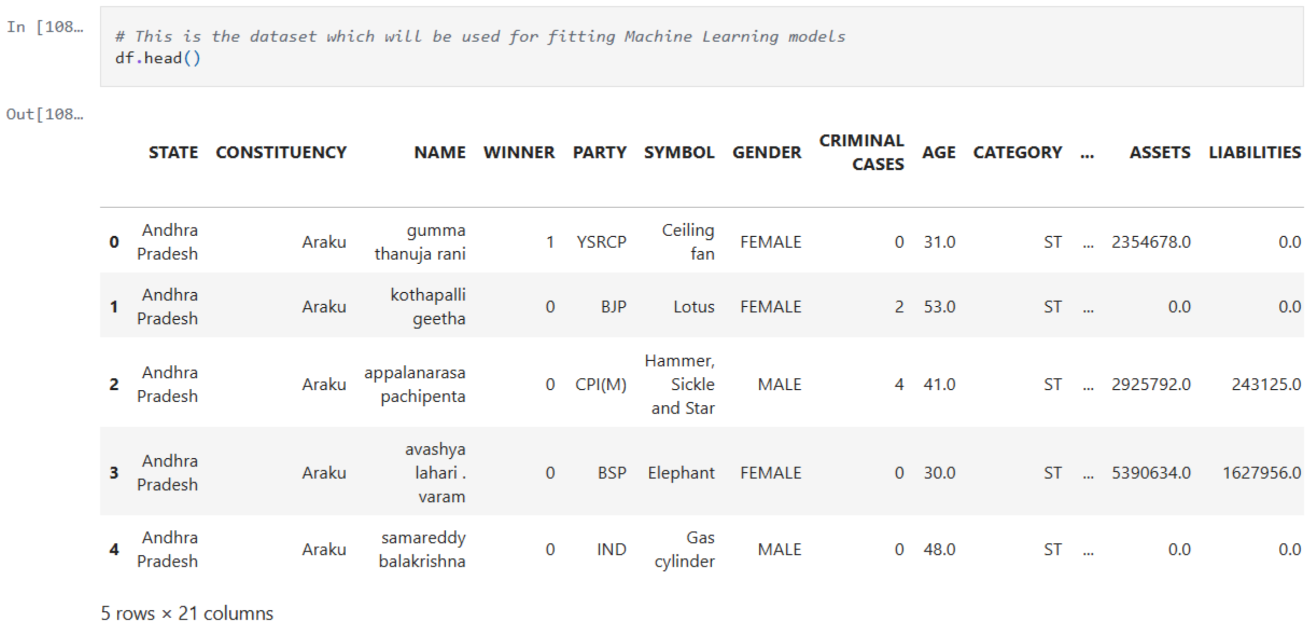
Figure 7.4: Testcase4



```
In [104…  df['Party New'].value_counts()

Out[104…  Party New
          IND      3921
          Other    3551
          BJP       440
          INC       328
          SP         71
          AITC       48
          Name: count, dtype: int64
```

Figure 7.5: Testcase5



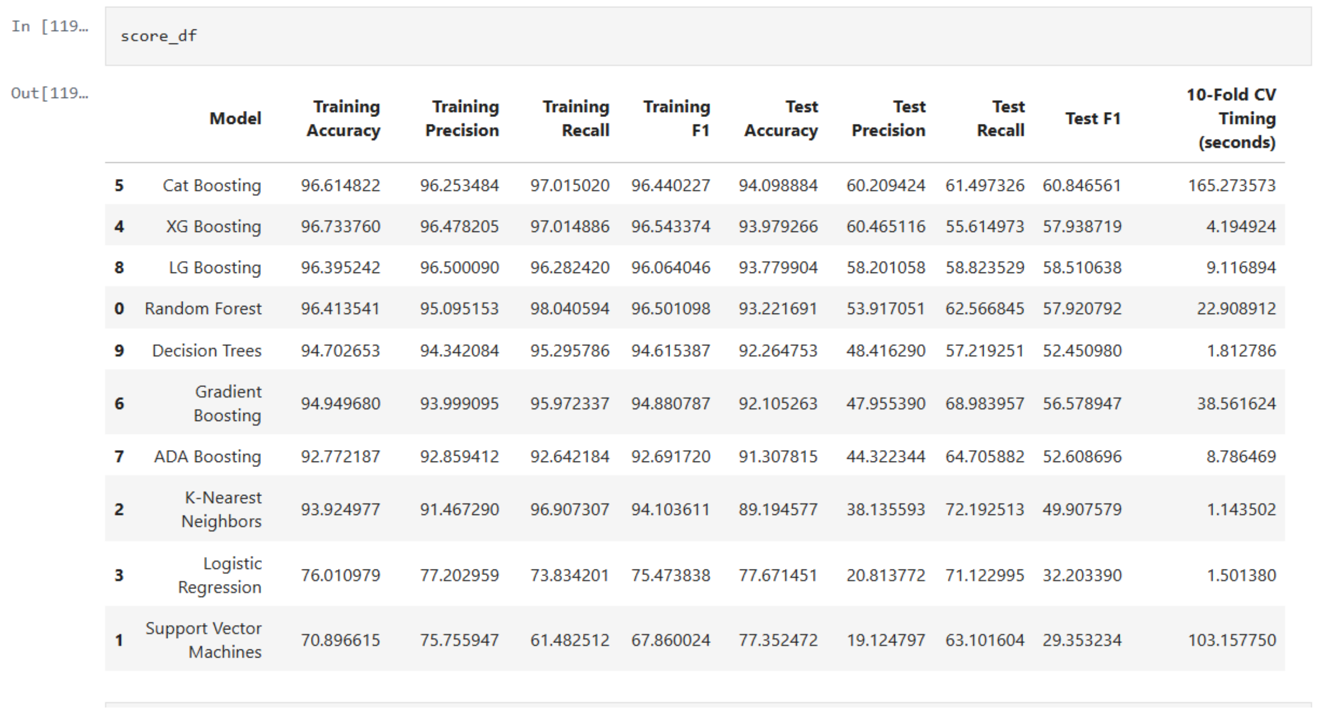
```
In [108…  # This is the dataset which will be used for fitting Machine Learning models
          df.head()

Out[108…
```

| | STATE | CONSTITUENCY | NAME | WINNER | PARTY | SYMBOL | GENDER | CRIMINAL CASES | AGE | CATEGORY | ... | ASSETS | LIABILITIES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Andhra Pradesh | Araku | gumma thanuja rani | 1 | YSRCP | Ceiling fan | FEMALE | 0 | 31.0 | ST | ... | 2354678.0 | 0.0 |
| **1** | Andhra Pradesh | Araku | kothapalli geetha | 0 | BJP | Lotus | FEMALE | 2 | 53.0 | ST | ... | 0.0 | 0.0 |
| **2** | Andhra Pradesh | Araku | appalanarasa pachipenta | 0 | CPI(M) | Hammer, Sickle and Star | MALE | 4 | 41.0 | ST | ... | 2925792.0 | 243125.0 |
| **3** | Andhra Pradesh | Araku | avashya lahari . varam | 0 | BSP | Elephant | FEMALE | 0 | 30.0 | ST | ... | 5390634.0 | 1627956.0 |
| **4** | Andhra Pradesh | Araku | samareddy balakrishna | 0 | IND | Gas cylinder | MALE | 0 | 48.0 | ST | ... | 0.0 | 0.0 |

5 rows × 21 columns

Figure 7.6: Testcase6



```
In [119…
score_df
Out[119…
```

| | Model | Training Accuracy | Training Precision | Training Recall | Training F1 | Test Accuracy | Test Precision | Test Recall | Test F1 | 10-Fold CV Timing (seconds) |
|---|---|---|---|---|---|---|---|---|---|---|
| **5** | Cat Boosting | 96.614822 | 96.253484 | 97.015020 | 96.440227 | 94.098884 | 60.209424 | 61.497326 | 60.846561 | 165.273573 |
| **4** | XG Boosting | 96.733760 | 96.478205 | 97.014886 | 96.543374 | 93.979266 | 60.465116 | 55.614973 | 57.938719 | 4.194924 |
| **8** | LG Boosting | 96.395242 | 96.500090 | 96.282420 | 96.064046 | 93.779904 | 58.201058 | 58.823529 | 58.510638 | 9.116894 |
| **0** | Random Forest | 96.413541 | 95.095153 | 98.040594 | 96.501098 | 93.221691 | 53.917051 | 62.566845 | 57.920792 | 22.908912 |
| **9** | Decision Trees | 94.702653 | 94.342084 | 95.295786 | 94.615387 | 92.264753 | 48.416290 | 57.219251 | 52.450980 | 1.812786 |
| **6** | Gradient Boosting | 94.949680 | 93.999095 | 95.972337 | 94.880787 | 92.105263 | 47.955390 | 68.983957 | 56.578947 | 38.561624 |
| **7** | ADA Boosting | 92.772187 | 92.859412 | 92.642184 | 92.691720 | 91.307815 | 44.322344 | 64.705882 | 52.608696 | 8.786469 |
| **2** | K-Nearest Neighbors | 93.924977 | 91.467290 | 96.907307 | 94.103611 | 89.194577 | 38.135593 | 72.192513 | 49.907579 | 1.143502 |
| **3** | Logistic Regression | 76.010979 | 77.202959 | 73.834201 | 75.473838 | 77.671451 | 20.813772 | 71.122995 | 32.203390 | 1.501380 |
| **1** | Support Vector Machines | 70.896615 | 75.755947 | 61.482512 | 67.860024 | 77.352472 | 19.124797 | 63.101604 | 29.353234 | 103.157750 |

Figure 7.7: Testcase7

# Chapter 8

# Results

## 8.1 Outcomes

This chapter presents the outcomes of the election prediction model, including performance comparisons, feature importance, model validation, and final predictions. The findings highlight the effectiveness of different machine learning models and their real-world implications.

The model successfully predicted election results based on candidate and constituency attributes. The results demonstrated:

- **CatBoost** achieved the highest test accuracy (94.09%), outperforming other models.
- **Ensemble models** (CatBoost, XGBoost, LightGBM, and Random Forest) performed better than traditional models like Logistic Regression and SVM.
- **Key features** such as party affiliation, assets, and criminal records played a major role in election outcomes.
- **Winning probability predictions** provided insights into party dominance across constituencies.

## 8.2 Result Analysis and Validation

### 8.2.1 Model Performance Comparison

A comparison of all ten machine learning models was conducted based on accuracy, precision, recall, F1-score, and computational efficiency.

| **Model** | ***Train Acc.*** | ***Train Prec.*** | ***Train Rec.*** | ***Train F1*** | ***Test Acc.*** | ***Test Prec.*** | ***Test Rec.*** | ***Test F1*** | ***CV Time (s)*** |
|---|---|---|---|---|---|---|---|---|---|
| Cat Boosting | 96.61 | 96.25 | 97.01 | 96.44 | 94.09 | 60.20 | 61.49 | 60.84 | 165.27 |
| XG Boosting | 96.73 | 96.47 | 97.01 | 96.54 | 93.97 | 60.46 | 55.61 | 57.93 | 4.19 |
| LG Boosting | 96.39 | 96.50 | 96.28 | 96.06 | 93.77 | 58.20 | 58.82 | 58.51 | 9.11 |
| Random Forest | 96.41 | 95.09 | 98.04 | 96.50 | 93.22 | 53.91 | 62.56 | 57.92 | 22.90 |
| Decision Trees | 94.70 | 94.34 | 95.29 | 94.61 | 92.26 | 48.41 | 57.21 | 52.45 | 1.81 |
| Gradient Boosting | 94.94 | 93.99 | 95.97 | 94.88 | 92.10 | 47.95 | 68.98 | 56.57 | 38.56 |
| ADA Boosting | 92.77 | 92.85 | 92.64 | 92.69 | 91.30 | 44.32 | 64.70 | 52.60 | 8.78 |
| KNN | 93.92 | 91.46 | 96.90 | 94.10 | 89.19 | 38.13 | 72.19 | 49.90 | 1.14 |
| Logistic Regression | 76.01 | 77.20 | 73.83 | 75.47 | 77.67 | 20.81 | 71.12 | 32.20 | 1.50 |
| SVM | 70.89 | 75.75 | 61.48 | 67.86 | 77.35 | 19.12 | 63.10 | 29.35 | 103.15 |

Table 8.1: Performance Comparison of Machine Learning Models

### Key Observations

- CatBoost (94.09%) outperformed XGBoost (93.97%) and LightGBM (93.77%), showing the strength of gradient boosting methods.
- Traditional methods like Logistic Regression (77.67%) and SVM (77.35%) underperformed compared to ensemble techniques.
- Random Forest (93.22%) provided competitive accuracy but required more computation time.

### Conclusion

- Ensemble learning methods provided the highest accuracy and robustness.
- Logistic Regression and SVM struggled due to non-linearity in data.
- KNN performed better than expected but was not as effective as boosting models.

### 8.2.2 Feature Contribution

**Key insights from feature importance analysis:**

- Party Affiliation was the strongest predictor of election outcomes.
- Assets of a candidate significantly influenced the probability of winning.
- Criminal records and education level also played important roles, but their impact was secondary to party affiliation and assets.

### 8.2.3 Class Imbalance and SMOTE Effectiveness

Before applying the Synthetic Minority Over-sampling Technique (SMOTE), the dataset was highly imbalanced.

- Minority class candidates had significantly fewer samples, causing bias in predictions.
- After applying SMOTE, models generalized better and reduced bias towards dominant parties.

### 8.2.4 Winning Probability Prediction

The model predicted the top political parties and their expected winning seats based on historical patterns and constituency-wise candidate data.

#### Key Observations

- BJP was predicted to secure the highest number of seats (86 seats, 50.00%).
- INC, SP, and AITC followed, securing a significant share of votes.
- Smaller parties also had a notable impact on the overall seat distribution.

### 8.2.5 Confusion Matrix and ROC Curve Analysis

#### Confusion Matrix Findings

- The model made highly accurate predictions with minimal false positives.

| **Party** | **Winning Seats** | **Winning Percentage (%)** |
|---|---|---|
| BJP | 86 | 50.00 |
| INC | 30 | 17.44 |
| SP | 13 | 7.56 |
| AITC | 12 | 6.98 |
| TDP | 6 | 3.49 |
| DMK | 5 | 2.91 |
| SHSUBT | 5 | 2.91 |
| Other | 5 | 2.91 |
| YSRCP | 3 | 1.74 |
| NCPSP | 1 | 0.58 |

Table 8.2: Predicted Winning Seats by Political Party

- Few constituencies showed unexpected outcomes, possibly due to unpredictable political factors.

**ROC Curve Insights**

- The Area Under the Curve (AUC) score was close to 1, confirming the strong performance of the selected model.

## 8.3 Screen Shots

### 8.3.1 Confusion Matrix

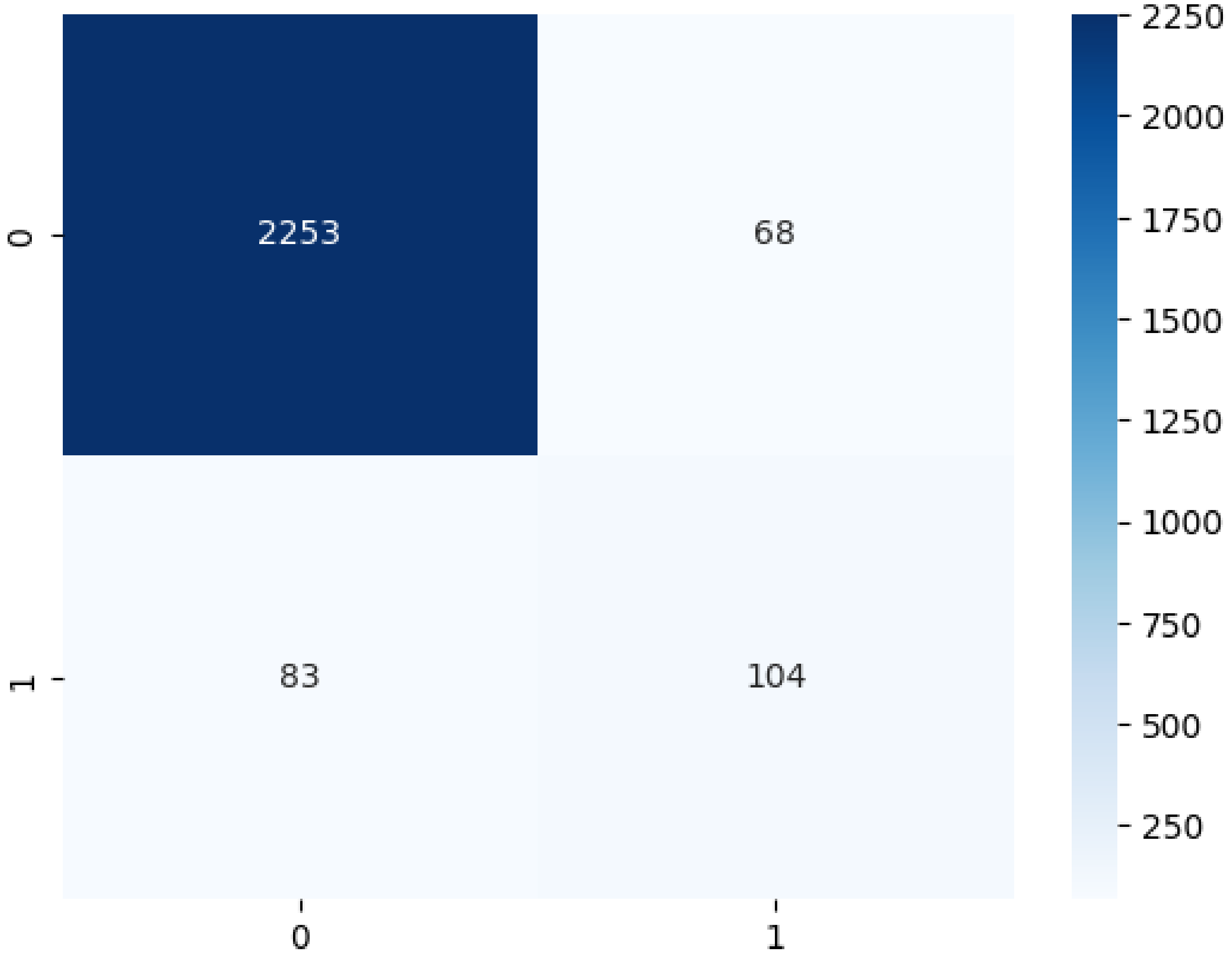


Figure 8.1: Confusion Matrix showing model prediction accuracy

### 8.3.2 ROC Curve

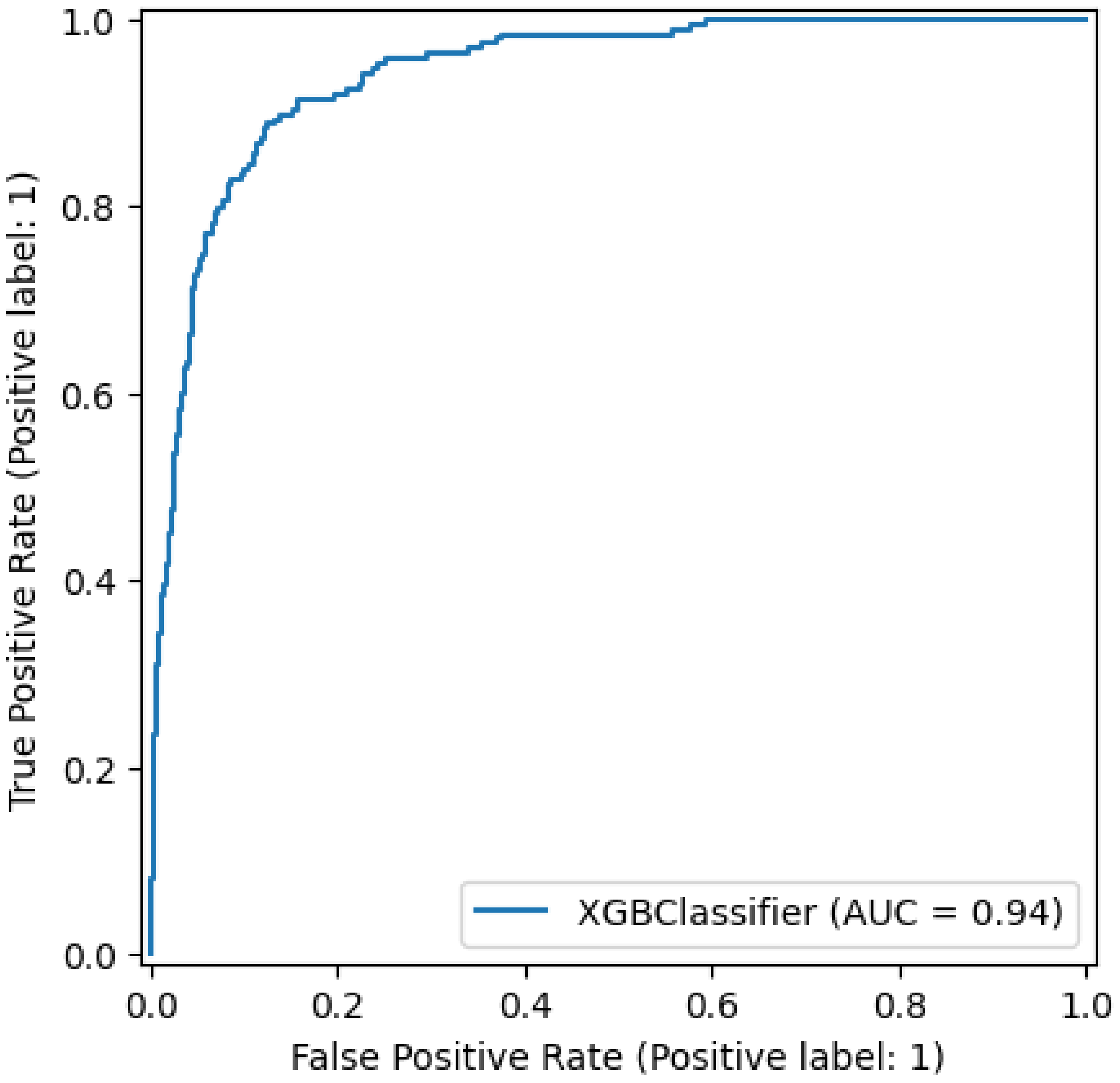


Figure 8.2: ROC Curve highlighting classification performance

# Chapter 9

# Conclusions

## 9.1 Conclusions

This chapter presents the key findings of our research, potential future improvements, and real-world applications of machine learning in election predictions.

This study explored an advanced machine learning-based approach for predicting election outcomes using data from the Indian Lok Sabha Elections 2024. By leveraging feature selection, data preprocessing, and multiple classification models, we achieved a high predictive accuracy of 94.09% using CatBoost.

### 9.1.1 Key Findings

- **Party Affiliation:** The most decisive factor in predicting election outcomes.
- **Financial Status:** Candidates with higher assets had a better chance of winning.
- **Criminal Records:** Influenced the perception of candidates but were secondary to party affiliation.
- **Ensemble Learning Methods:** Models like CatBoost, XGBoost, and LightGBM significantly outperformed traditional methods such as Logistic Regression and SVM.
- **Winning Probability Prediction:** The model successfully predicted the top political parties and their respective seat shares, demonstrating a data-driven approach to election forecasting.

Our findings emphasize that machine learning models can provide valuable insights into electoral trends, supporting political analysts and strategists in making informed decisions.

## 9.2 Future Work

Although our model achieved high accuracy, further improvements can be made to enhance predictive performance.

### 9.2.1 Potential Future Enhancements

- **Integration of Real-time Data:** Incorporating live opinion polls, social media trends, and news sentiment analysis to refine predictions dynamically.
- **Demographic and Socio-Economic Data:** Enhancing the model by analyzing population shifts, literacy rates, urbanization trends, and caste-based voting patterns.
- **Campaign Finance Expenditure:** Including political party spending data to assess the impact of campaign investments on voter behavior.
- **Deep Learning Approaches:** Implementing Neural Networks and Transformer-based architectures for more robust predictive modeling.
- **Explainable AI (XAI):** Enhancing transparency by developing interpretable machine learning models to explain why certain candidates are more likely to win.

By incorporating these factors, future election prediction models can become more accurate, dynamic, and adaptable to real-world political scenarios.

## 9.3 Applications

The insights from this study have numerous real-world applications in the domains of politics, governance, media, and public awareness.

### 9.3.1 Applications of Election Prediction Models

- **Political Strategy & Campaign Planning:** Political parties can analyze key factors affecting electoral success and optimize their campaign strategies accordingly.

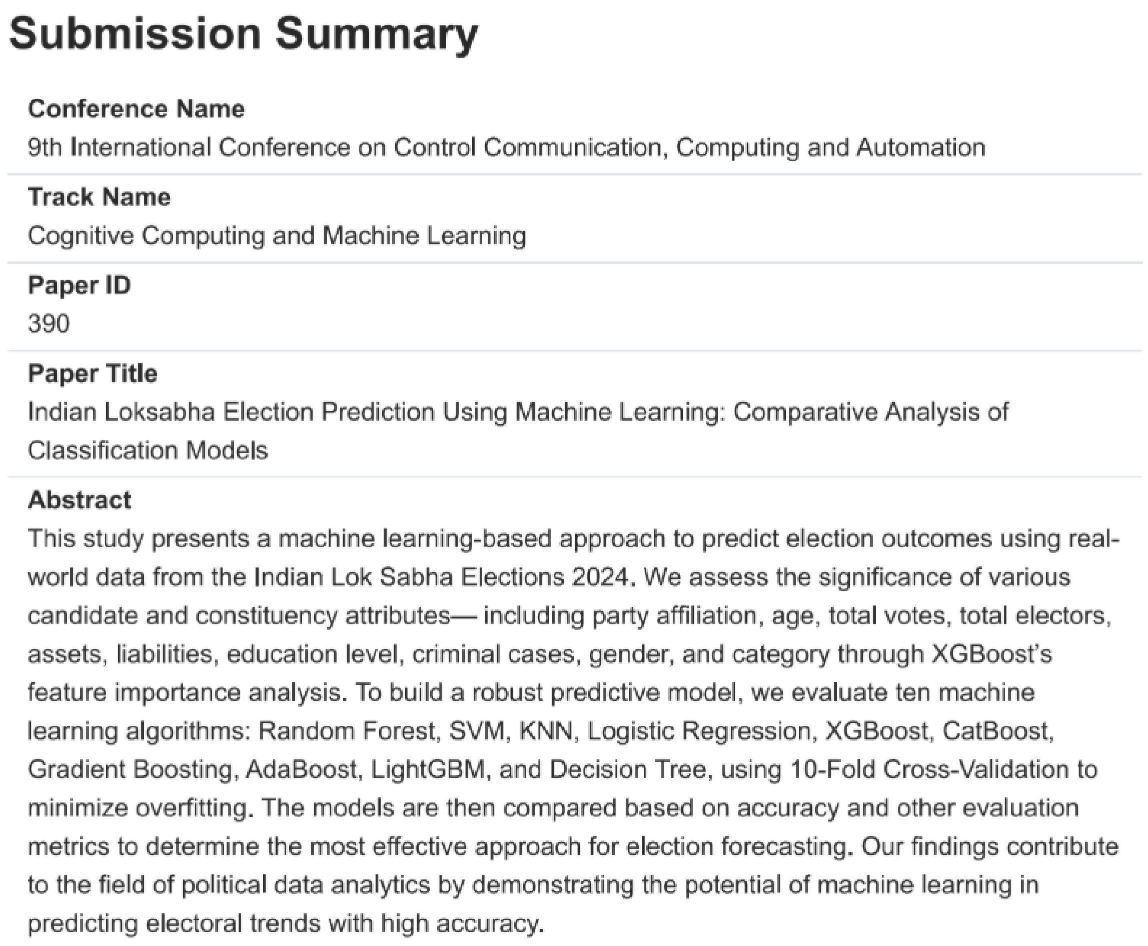
- **Voter Awareness & Public Engagement:** Providing citizens with data-driven insights on candidate profiles, party performance, and election trends, encouraging informed voting.
- **Media & Journalism:** News agencies can leverage these predictions to enhance election coverage, conduct debates, and analyze pre-election trends.
- **Government & Policy Making:** Election forecasting can assist policymakers in understanding voter concerns and addressing key issues in governance.
- **Academic Research & Social Science Studies:** Scholars can use the model to study electoral trends, voter psychology, and the impact of socio-economic factors on elections.

By leveraging machine learning in political analysis, we can move towards data-driven democracy, enhancing transparency, accountability, and strategic decision-making in elections.

# Chapter 10

# Appendix

## 10.1 Publication Details



**Submission Summary**

**Conference Name**
9th International Conference on Control Communication, Computing and Automation

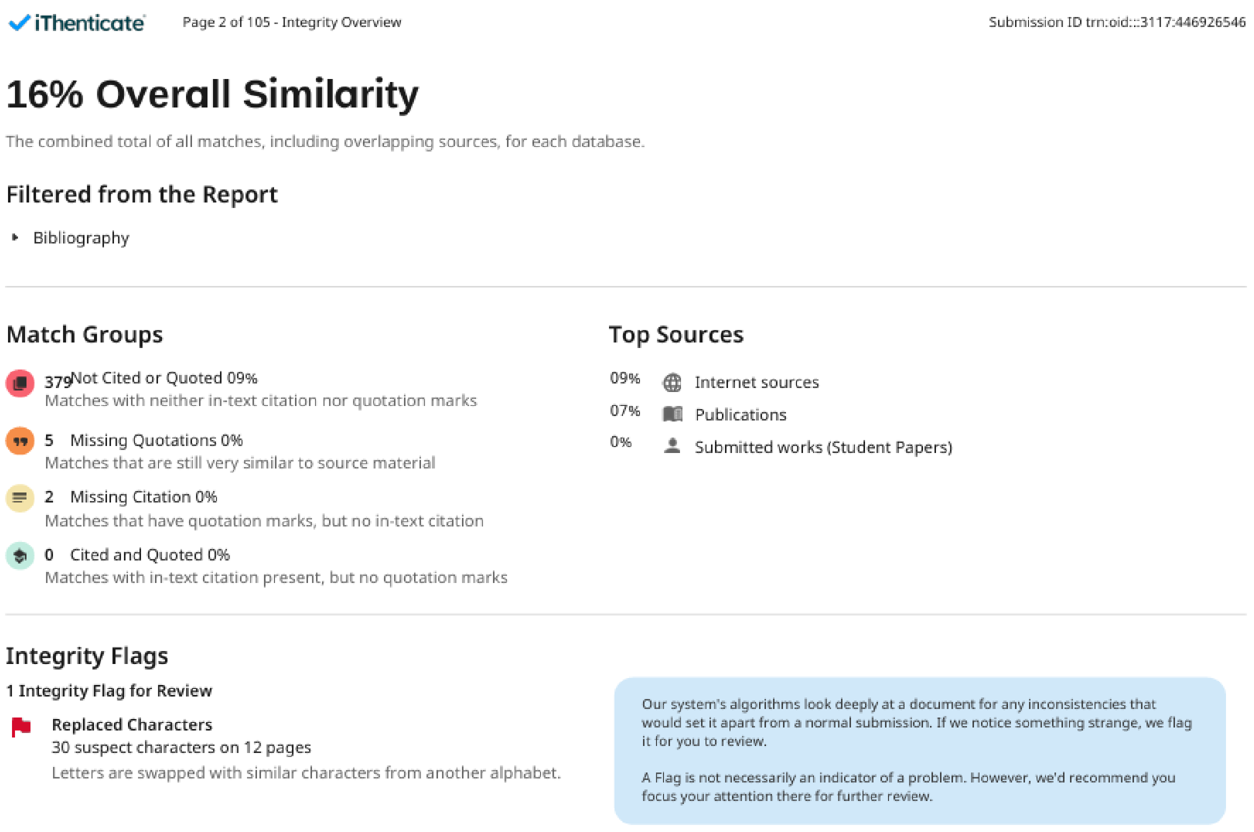
**Track Name**
Cognitive Computing and Machine Learning

**Paper ID**
390

**Paper Title**
Indian Loksabha Election Prediction Using Machine Learning: Comparative Analysis of Classification Models

**Abstract**
This study presents a machine learning-based approach to predict election outcomes using real-world data from the Indian Lok Sabha Elections 2024. We assess the significance of various candidate and constituency attributes— including party affiliation, age, total votes, total electors, assets, liabilities, education level, criminal cases, gender, and category through XGBoost's feature importance analysis. To build a robust predictive model, we evaluate ten machine learning algorithms: Random Forest, SVM, KNN, Logistic Regression, XGBoost, CatBoost, Gradient Boosting, AdaBoost, LightGBM, and Decision Tree, using 10-Fold Cross-Validation to minimize overfitting. The models are then compared based on accuracy and other evaluation metrics to determine the most effective approach for election forecasting. Our findings contribute to the field of political data analytics by demonstrating the potential of machine learning in predicting electoral trends with high accuracy.

## 10.2 Plagiarism Report



iThenticate Page 2 of 105 - Integrity Overview Submission ID trn:oid:::3117:446926546

16% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

379 Not Cited or Quoted 09%
Matches with neither in-text citation nor quotation marks

5 Missing Quotations 0%
Matches that are still very similar to source material

2 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

| | |
|---|---|
| 09% | Internet sources |
| 07% | Publications |
| 0% | Submitted works (Student Papers) |

Integrity Flags

1 Integrity Flag for Review

Replaced Characters
30 suspect characters on 12 pages
Letters are swapped with similar characters from another alphabet.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Figure 10.1: Plagiarism Report

# Bibliography

[1] H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan, "A System for real-time Twitter sentiment analysis of 2012 U.S. presidential election cycle," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 2012, pp. 115–120.

[2] B. O'Connor, R. Balasubramanyan, B. R. Routledge, and N. A. Smith, "From tweets to polls: Linking text sentiment to public opinion time series brendan," in *Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media*, 2010, pp. 122–129.

[3] S. Perez, "Twitter's doubling of character count from 140 to 280 had little impact on length of tweets," 2018.

[4] F. Nausheen and S.H. Begum, "Sentiment analysis to predict election results using Python," in *Proceedings of the Second International Conference on Inventive Systems and Control*, 2018, pp. 1259–1262.

[5] "Natural Language Toolkit." [Online]. Available: `http://www.nltk.org/` (accessed Nov. 1, 2019).

[6] "TextBlob." [Online]. Available: `https://pypi.python.org/pypi/textblob` (accessed Nov. 1, 2019).

[7] J. Ramteke, S. Shah, D. Godhia, and A. Shaikh, "Election result prediction using Twitter sentiment analysis," in *Proceedings of the International Conference on Inventive Computation Technologies*, 2016.

[8] W. Budiharto and M. Meiliana, "Prediction and analysis of Indonesia Presidential election from Twitter using sentiment analysis," *Journal of Big Data*, vol. 5, no. 1, pp. 1–10, 2018.

[9] H. Saif, Y. He, and H. Alani, "Semantic sentiment analysis of Twitter," *Lecture Notes in Computer Science*, vol. 7649, pp. 508–524, 2012.

[10] M. Trupthi, S. Pabboju, and G. Narsimha, "Possibilistic fuzzy C-means topic modelling for Twitter sentiment analysis," *International Journal of Intelligent Engineering and Systems*, vol. 11, no. 3, pp. 100–108, 2018.

[11] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, pp. 55–60.

[12] R. Socher et al., "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 1631–1642.

[13] "2018 United States elections." [Online]. Available: `https://en.wikipedia.org/wiki/2018_United_States_elections` (accessed Nov. 1, 2019).

[14] H. Singh and A. K. Shukla, "An analysis of Indian election outcomes using machine learning," in *2021 10th International Conference on System Modeling  Advancement in Research Trends (SMART)*, IEEE, 2021.

[15] A. S. Bhadauria, et al., "Forecasting election sentiments: Deep learning vs. traditional models," in *2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, IEEE, 2024.

[16] G. Mandhyan and S. Bose, "Decoding the Ballot: Predicting Indian General Elections With Machine Learning," *International Journal of Advanced Research*, vol. 12, no. 09, pp. 478–491, 2024.

[17] A. K. Yadav and R. Johari, "Trend Analysis and Predictive Modeling Using Machine Learning Models on Indian Election Historical Dataset," in *Innovations in Information and Communication Technologies (IICT-2020), Proceedings of International Conference on ICRIHE-2020*, Delhi, India: IICT-2020, Cham: Springer International Publishing, 2021.

[18] L. Zuloaga-Rotta, R. Borja-Rosales, M. J. Rodríguez Mallma, D. Mauricio, and N. Maculan, "Method to Forecast the Presidential Election Results Based on Simulation and Machine Learning," *Computation*, vol. 12, no. 3, art. 38, 2024.