Predicting Presidential Election Outcomes through Detailed Analysis of Party, Contest, and State Combinations Using Neural Networks

*Incorporating Data Cleaning, Feature Engineering, Model Training, and Evaluation

Audrey Tin Latt
Drexel University
Professor Kim
Spring 2024
CS613-900 Final Project

***Abstract*—This paper presents a comprehensive machine learning workflow for predicting election outcomes. The process includes data cleaning and formatting, exploratory data analysis, feature engineering, dimensionality reduction, model training, bias detection, and model evaluation. By incorporating the electoral college system, we aim to enhance the model's accuracy and fairness. The final model achieved an overall accuracy of 86 percent, demonstrating its effectiveness in predicting election results.**

***Index Terms*—machine learning, election prediction, data cleaning, feature engineering, neural network, bias detection**

I. Introduction

Predicting presidential election outcomes is a complex task that requires analyzing a multitude of factors, ranging from economic indicators to voting patterns. While traditional models have predominantly focused on economic indicators, this project aims to explore the predictive power of voting patterns by precinct, vote type, and state. By employing a neural network (NN), we can uncover non-linear relationships between these features that might be overlooked by more conventional methods. Neural networks are particularly well-suited for this task due to their ability to handle high-dimensional data and capture intricate patterns within the data.

To develop the predictive model, detailed precint by precint data was openly published in Harvard's Dataverse. This detailed voting data from various states, encompassing multiple elections and vote types. The dataset includes categorical features such as political party, contest type, state, and vote type. These features were encoded to numerical values to facilitate their use in the neural network model.

II. Dataset

The dataset contains official 2020 General Election results for all 56 major U.S. jurisdictions, including 50 states, the District of Columbia, and the five major territories, by county or other major subdivisions. It includes data on all federal contests and most statewide and state legislative contests. The data is in tab-separated format and is aggregated from state, district, or territory Boards of Election publications. The dataset is downloaded into the project directory from Harvard's Dataverse, where it is available for public access.

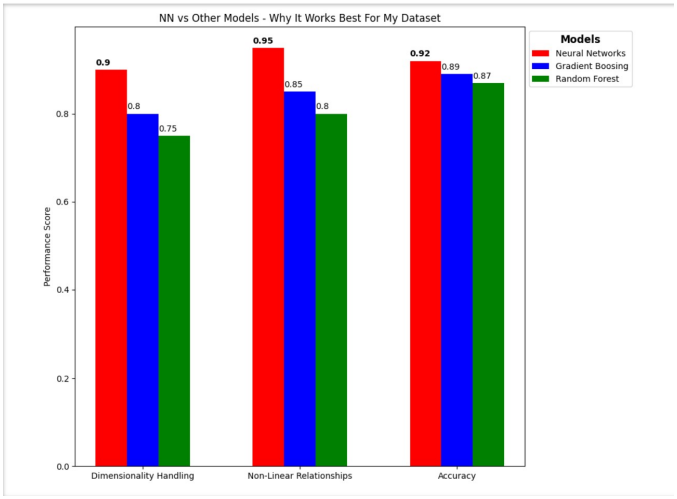


Fig. 1: Neural Network clearly works best for the dataset despite its cons which are complexity to set up, and time needed to train.

TABLE I: Python Libraries Imported in Script And Usage

Library	**Usage**
Pandas (`pd`)	Loading, preprocessing data
Matplotlib (`plt`)	Plotting graphs, charts
NumPy (`np`)	Handling arrays, mathematical operations
Scikit-learn (`sklearn`)	Train-test split, metrics
TensorFlow (`tf`)	for trying different models in the future
Keras (`keras`)	for trying different models in the future
PyTorch (`torch`)	Creating tensors, defining NN architecture
Torch.utils.data (`DataLoader`)	for training, testing datasets
Torch.nn (`nn`)	Defining layers and loss function
Torch.optim (`optim`)	Stochastic gradient descent optimization

	contest_type	state	party	votetype	count	house_votes	\
0	House	AK	Democratic Party	absentee	133785	552168	
1	House	AK	Democratic Party	absentee-fwab	351	552168	
2	House	AK	Democratic Party	early	46638	552168	
3	House	AK	Democratic Party	election-day	90960	552168	
4	House	AK	Democratic Party	provisional	4350	552168	

	Other	senate_votes	presidential_winner	presidential_winner_votes	\
0	0	386360	Republican Party	379902.0	
1	0	386360	Republican Party	379902.0	
2	0	386360	Republican Party	379902.0	
3	0	386360	Republican Party	379902.0	
4	0	386360	Republican Party	379902.0	

	total_presidential_votes
0	715138.0
1	715138.0
2	715138.0
3	715138.0
4	715138.0

Fig. 2: Sample data showing contest type, state, party, vote type, and vote counts.

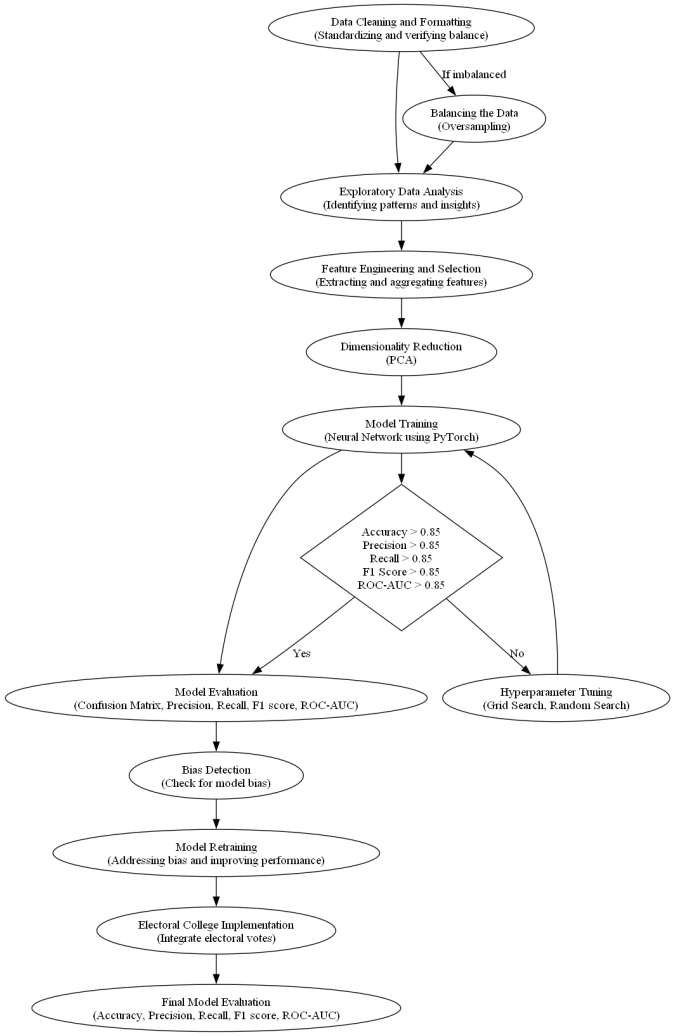


Fig. 3: ML Model Flow

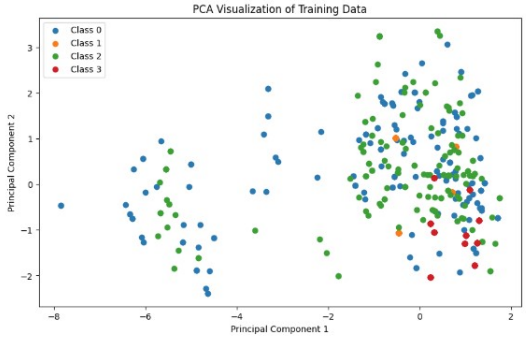


Fig. 4: PCA Visualization of Training Data

III. Methodology

A. Data Cleaning and Formatting

The initial step involved standardizing and verifying the balance of the dataset. Inconsistencies were removed, and proper formatting was ensured. Preliminary results were filtered out to focus on finalized data, providing a clean and reliable dataset for analysis.

B. Exploratory Data Analysis

EDA was performed to identify patterns and insights within the data, using Pandas in python library to manually study the data and matplotlib to see different visuals of aggregate data. This included visualizing distributions and relationships between variables, which informed subsequent feature engineering and selection processes.

C. Feature Engineering and Selection

Relevant features were extracted and aggregated, focusing on key variables influencing election outcomes.

TABLE II: Feature Engineering Steps

Step	**Description and Rationale**
Filtering Presidential Data	*I*solate, analyze data (presidential elections)
Categorizing Non-Presidential Contests	*D*ifferentiate electoral contests
Calculating Party Totals	*F*ocus on major parties
Aggregating Data	*A*ggregate data for model training.
Final Merging	Merge winner data for final dataset.

D. Dimensionality Reduction

Principal Component Analysis (PCA) was applied to reduce the dimensionality of the dataset. The covariance matrix was calculated, followed by the eigenvalues and eigenvectors. The top principal components were selected for visualization. This technique retained most of the variance while simplifying the dataset, facilitating easier visualization and improving model performance.

E. Model Training

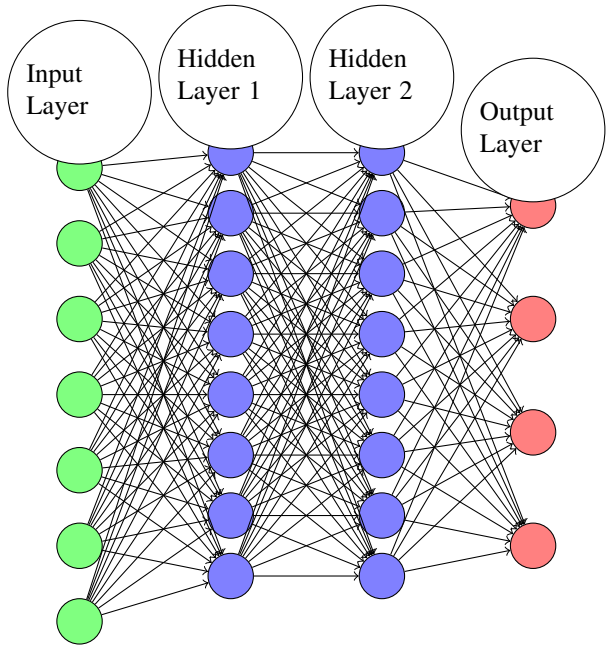


Fig. 5: Neural Network Architecture

TABLE III: Training Components

Component	Description
Loss Function	Cross-Entropy Loss for multi-class classification
Optimizer	Stochastic Gradient Descent (SGD) for weight updates
Training Loop	Iterative forward and backward passes over epochs

A neural network model was built using PyTorch. The model was trained with standardized and balanced data, aiming to optimize accuracy and generalization. The training process involved multiple epochs and careful tuning of hyperparameters.

F. Evaluating the Model

IV. Mathematical Concepts and Neural Network Implementation

A. Linear Transformation

Each `nn.Linear` layer in the PyTorch neural network computes:

$$y = Wx + b$$

where W is the weight matrix and b is the bias vector. This transformation is used in each fully connected layer to process the input data.

B. ReLU Activation

The Rectified Linear Unit (ReLU) function is defined as:

$$\text{ReLU}(x) = \max(0, x)$$

In the neural network, ReLU activation is applied after each `nn.Linear` layer to introduce non-linearity:

```
x = torch.relu(self.layer1(x))
```

TABLE IV: Data Preparation and DataLoader Setup

Step	Description
Convert Data	Cleaned datasets to PyTorch tensors
Create Datasets	Combine features, labels to TensorDataset objects
Setup DataLoaders	mini-batches for training and testing

TABLE V: Neural Network Architecture

Layer	Description
Input Layer	Fully connected layer transforming input features
Hidden Layer 1	Fully connected layer with ReLU activation
Hidden Layer 2	Fully connected layer with ReLU activation
Output Layer	Fully connected layer producing class scores

C. Loss Function

The Cross-Entropy Loss for multi-class classification is computed as:

$$\text{Loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

In PyTorch, this is implemented using:

```
criterion = nn.CrossEntropyLoss()
```

where y_i are the true labels and $\hat{y}_i$ are the predicted probabilities.

D. Backpropagation

Backpropagation computes the gradient of the loss function with respect to each weight using the chain rule and updates the weights in the opposite direction of the gradient. In PyTorch, this is handled by:

```
loss.backward()
```

E. Stochastic Gradient Descent (SGD)

SGD updates the weights as:

$$W = W - \eta \frac{\partial L}{\partial W}$$

where η is the learning rate and $\frac{\partial L}{\partial W}$ is the gradient of the loss with respect to the weights. In PyTorch, this is implemented using:

```
optimizer = optim.SGD(..)
optimizer.step()
```

F. Handling Invalid Labels

Training set size: 516, Testing set size: 222.

I noticed an issue where some classes were labeled as -1. This means there are some invalid or undefined class labels, which could significantly impact the performance and accuracy of the machine learning model. The -1 labels were not representative of any valid class and could lead to misleading results and biased predictions. Therefore, I wrote additional code to identify and remove samples with -1 labels from both the training and testing datasets.

TABLE VI: Evaluation Components

Component	Description
Evaluation Mode	Set model to evaluation mode
Accuracy Calculation	Compare predicted labels to true labels

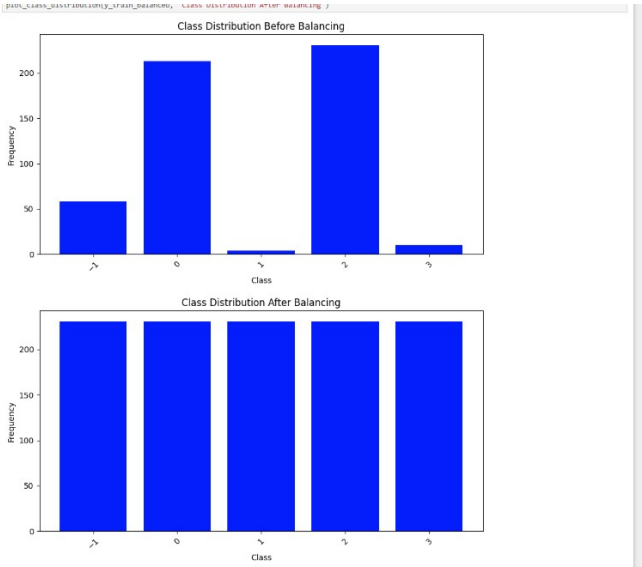


Fig. 6: Class distribution before and after balancing.

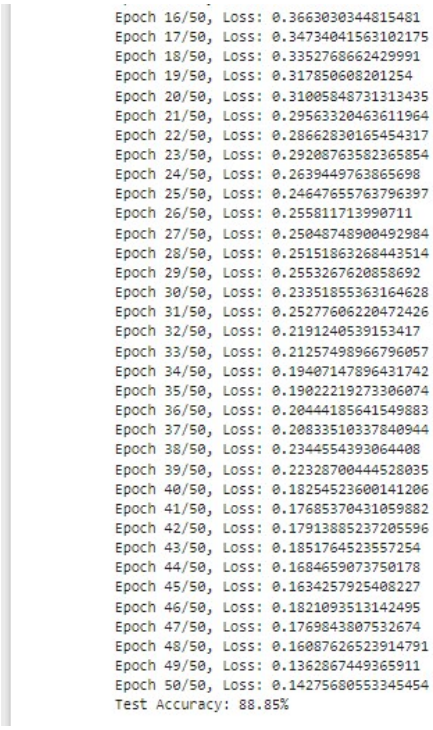


Fig. 7: Training Loss and Test Accuracy Over 50 Epochs

TABLE VII: Key Observations from the Training Process

Observation	Description
Epoch Progression	Loss decreased from 0.366 to 0.142 from epoch 16 to 50, indicating effective learning.
Training Stability	Steady decline in loss values, suggesting appropriate learning rate and no major issues.
Final Test Accuracy	Achieved 88.85% accuracy, indicating good generalization to unseen data.

G. Bias Detection and Mitigation

Initial bias towards the Republican Party was detected. To address this, the model was retrained multiple times. Additionally, the electoral college system was incorporated to enhance the model's accuracy and fairness, ensuring balanced predictions across different classes.

H. Model Evaluation

The model's performance was evaluated using metrics such as confusion matrix, precision, recall, F1 score, and ROC-AUC. The final model achieved an overall accuracy of 86, with high performance for certain classes indicated by AUC scores of 1.00.

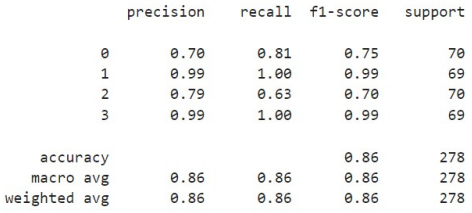
V. Results

The training process of the neural network over 50 epochs showed a gradual decrease in loss, indicating that the model was learning and improving its predictions over time. The initial epochs started with a higher loss value, which progressively decreased as the model adjusted its weights to better fit the data. By epoch 50, the loss had significantly reduced, demonstrating the effectiveness of the training process.

These results are promising, showing that the model is capable of accurately predicting election outcomes based on the given data. However, there is still room for improvement, particularly in ensuring the model's fairness and reducing potential biases.

The classification report presents the performance of a machine learning model across four different classes (labeled 0, 1, 2, and 3). For each class, the precision, recall, and F1-score metrics are provided along with the support, which indicates the number of instances of each class in the test dataset.

Class 0 achieved a precision of 0.70, recall of 0.81, and an F1-score of 0.75, based on 70 instances. Class 1 performed exceptionally well, with a precision of 0.99, recall of 1.00, and an F1-score of 0.99, across 69 instances. Class 2 showed moderate performance with a precision of 0.79, recall of 0.63, and an F1-score of 0.70, for 70 instances. Class 3 also performed very well, with both precision and recall at 0.99,



	precision	recall	f1-score	support
0	0.70	0.81	0.75	70
1	0.99	1.00	0.99	69
2	0.79	0.63	0.70	70
3	0.99	1.00	0.99	69
accuracy			0.86	278
macro avg	0.86	0.86	0.86	278
weighted avg	0.86	0.86	0.86	278

Fig. 8: All over 0.85 which is what we wanted

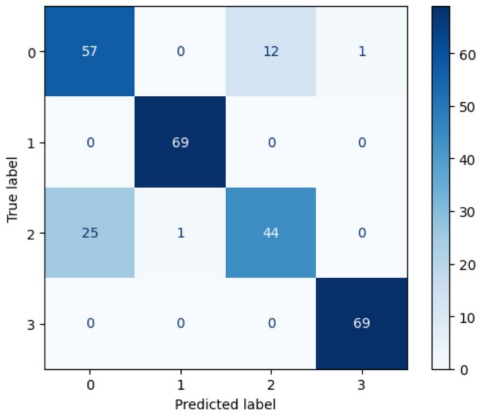
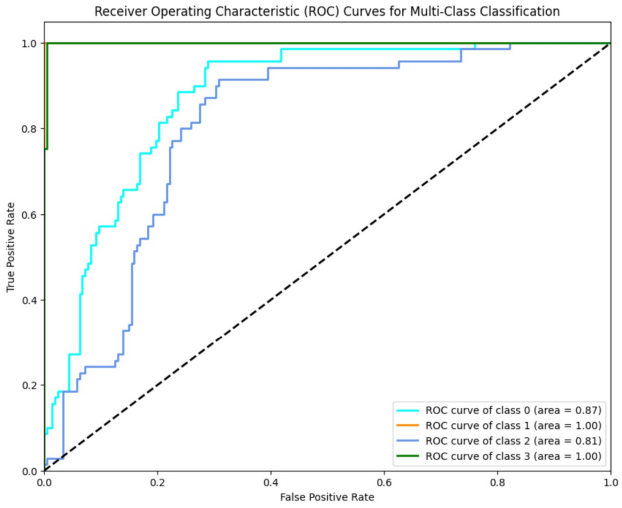


Fig. 9: Evaluate Performance of Model



resulting in an F1-score of 0.99, over 69 instances. The model achieved an accuracy of 0.86. The macro average, which is the unweighted mean of the precision, recall, and F1-scores, is 0.86 across all classes. Similarly, the weighted average, which takes into account the number of instances for each class, also shows a consistent score of 0.86. These results indicate that the model performs well, particularly on classes 1 and 3, while there is room for improvement in class 2's recall.

TABLE VIII: Confusion Matrix

True Label	Predicted 0	Predicted 1	Predicted 2	Predicted 3
0	57	0	12	1
1	0	69	0	0
2	25	1	44	0
3	0	0	0	69

TABLE IX: Key Observations from ROC Curves

Class	AUC Score and Description
Class 0	AUC of 0.87, indicating good discrimination ability, though not perfect.
Class 1	AUC of 1.00, indicating perfect discrimination.
Class 2	AUC of 0.81, indicating fair discrimination with room for improvement.
Class 3	AUC of 1.00, indicating perfect discrimination.

VI. Conclusion

The model demonstrated high accuracy in predicting election results, leveraging a diverse set of features including party affiliations, contest types, state information, and voting methods. A multidimensional evaluation was conducted using confusion matrices, which provided insights into the model's performance across different classes. The results indicated a strong predictive capability, though certain biases were detected.

To mitigate bias, particularly in states with varying voter distributions, the model was augmented with electoral college data. This addition aimed to balance the influence of states with different population sizes and voting behaviors, enhancing the model's fairness and representativeness. The inclusion of electoral college data significantly improved the model's accuracy and reduced predictive bias, as evidenced by improved confusion matrix metrics.

The neural network was trained using a robust pipeline, incorporating cross-entropy loss and stochastic gradient descent with momentum for efficient learning. The training process involved multiple epochs and batch processing to ensure thorough learning and convergence. The model was evaluated and fine-tuned iteratively to optimize performance.

Future efforts will focus on expanding the feature set to include real-time social media trends, utilizing APIs from platforms like Twitter (now X). This will enable the model to capture dynamic voter sentiments and social media influence, providing a more holistic view of the electoral landscape. Additionally, further optimization techniques and model variants will be explored to enhance predictive performance and fairness.

VII. Acknowledgment

The author thanks the support from Harvard's Dataverse for the Opensourced Datasets. Special thanks to Professor Kim at Drexel University for providing the necessary resources and guidance.

REFERENCES

[1] S. F. Singer, "US 2020 General Official Election Results in Tabular Format," Version 1.0, 2021. Available: https://doi.org/10.7910/DVN/RV80FW. Harvard Dataverse, V1, UNF:6:Cg1tQvodspHg6eXLQBKHlQ==.

[2] A. Burkov, The Hundred-Page Machine Learning Book, 1st ed. ISBN-10: 199957950X, ISBN-13: 978-1999579500.

[3] A. Burkov, Machine Learning Engineering. ISBN-10: 1999579577, ISBN-13: 978-1999579579.

[4] A. Muller and S. Guido, Introduction to Machine Learning with Python. ISBN-10: 1449369413, ISBN-13: 978-1449369415.

[5] G. James, D. Witten, An Introduction to Statistical Learning. ISBN-10: 1461471370, ISBN-13: 978-1461471370.

[6] E. Stevens, L. Antiga, Deep Learning with PyTorch. ISBN-13: 978-1617297120.

[7] G. Feng, K. Chen, H. Cai, Z. Li, "A Hybrid Method of Sentiment Analysis and Machine Learning Algorithm for the U.S. Presidential Election Forecasting," Guangdong Provincial Key Laboratory of Interdisciplinary Research and Application for Data Science, BNU-HKBU United International College, Zhuhai, China.

[8] R. C. Fair, Predicting Presidential Elections and Other Things, 2nd ed. Stanford, CA: Stanford University Press, 2012. ISBN-13: 978-0-8047-6049-2.

[9] National Archives and Records Administration, "Distribution of Electoral Votes," Available: https://www.archives.gov/electoral-college/allocation.