Stats 101C Final Classification Report

Group 8

By Tianlin Yue, Ziwei Guo, Rueiyu Chang, Kai Liang, Yiming Xue

August 3, 2024

1 Introduction

This project focuses on predicting the outcome of the 2020 US Presidential Election at the county level. The dataset includes 3,111 counties or county equivalents, with 75% (2,331 rows) designated as training data and 25% (780 rows) as test data. The primary goal is to predict whether Biden or Trump will win each county based on various demographic and educational information collected from the US Census Bureau. While due to data discrepancies, Alaska has been excluded from the analysis since its votes are reported in 41 districts rather than counties. Moreover, the research suggested demographic and behavioral factors, such as gender, race, education, and the impact of the COVID-19 pandemic has significantly influenced voting patterns in the 2020 election (Behind Biden's 2020 Victory, 2021). Overall, this report aims to analyze the provided data, develop predictive models, and evaluate the performance to accurately forecast the election results.

2 Data Analysis

To gain deeper insights into the data and understand the relationships between various features and the target variable, we will use visualizations. These visualizations help us to identify patterns, trends, and anomalies that may not be immediately apparent from the raw data or summary statistics alone.

The dataset was first cleaned by removing rows with missing values in key columns (x0001e, income_per_cap_2019, and income_category). Income categories were created by dividing the income per capita (2019) into four quantiles: Low, Medium, High, and Very High.

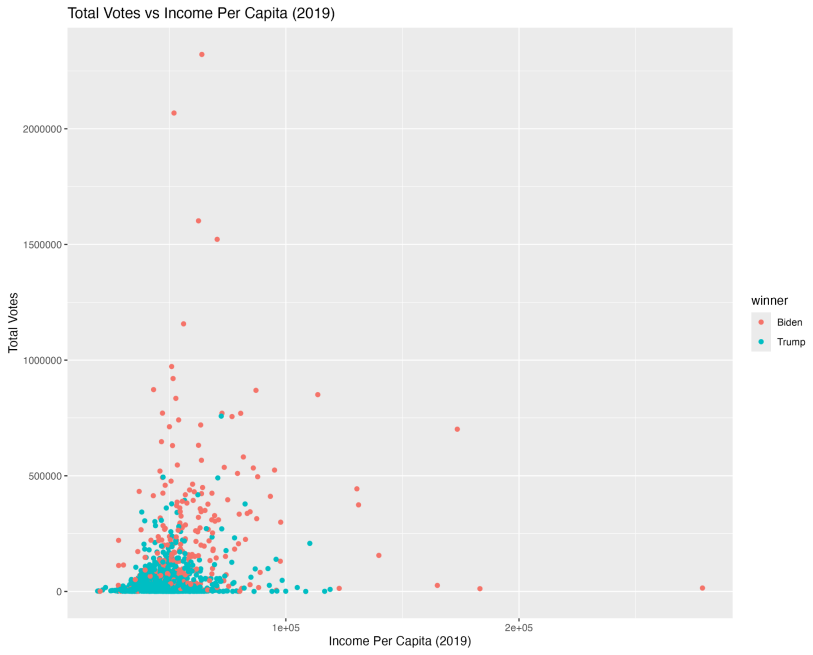


Figure 2.1

Figure 2.1 illustrates the relationship between income per capita (2019) and total votes. It shows that both parties are predominantly clustered in the lower income range, with a few outliers in the higher income range. This suggests that counties with lower income levels have higher total votes.

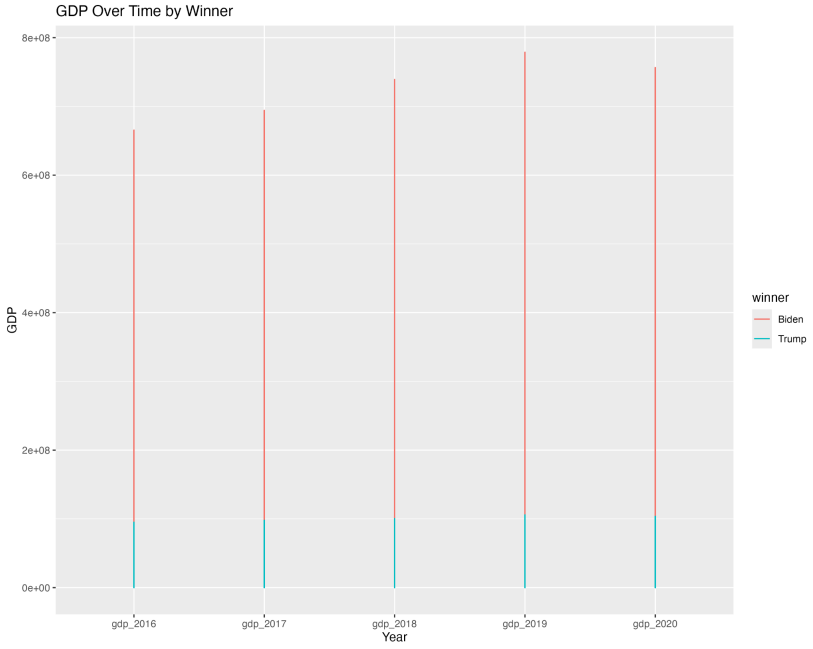


Figure 2.2

Figure 2.2 shows the GDP trends over time categorized by the election winner. The red lines represent counties won by Biden, while the blue lines represent counties won by Trump. Notably, there is a significant difference between the two groups, with Biden-winning counties showing markedly higher GDP in certain years.

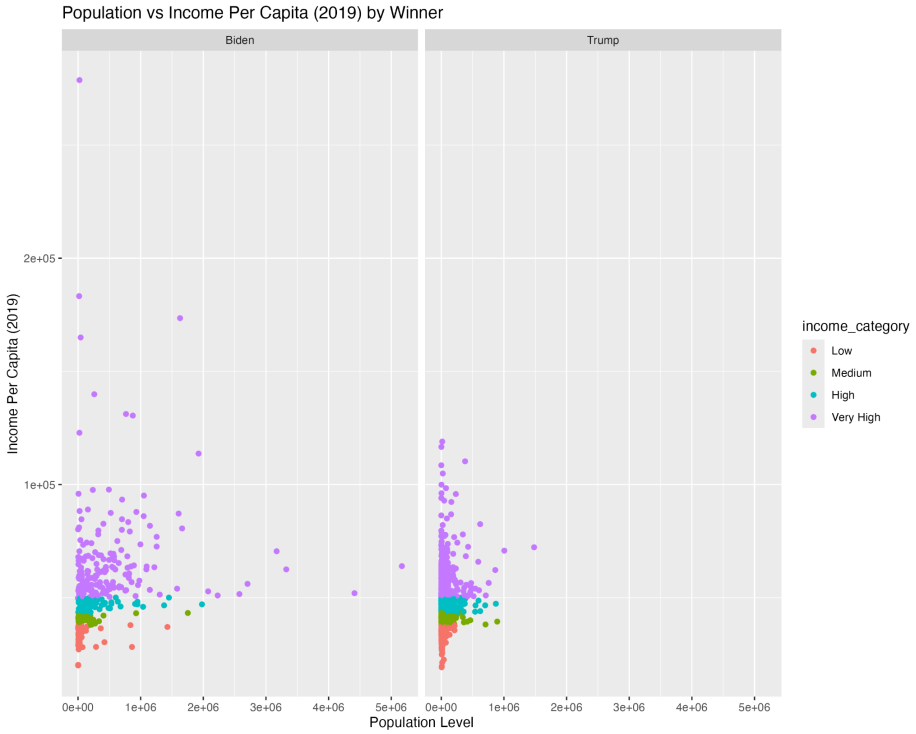


Figure 2.3

Figure 2.3 displays the relationship between population level and income per capita (2019), categorized by the election winner. The data is further divided into income categories: Low, Medium, High, and Very High. The facet wrap separates the data into two panels, one for Biden-winning counties and one for Trump-winning counties.

In both panels, there is a visible clustering of points in the lower population levels and income ranges, with some counties showing higher population levels and incomes. Biden-winning counties (left panel) exhibit a broader distribution in the higher population levels and income categories compared to Trump-winning counties (right panel). This suggests that counties with higher population levels and incomes are more likely to have voted for Biden.

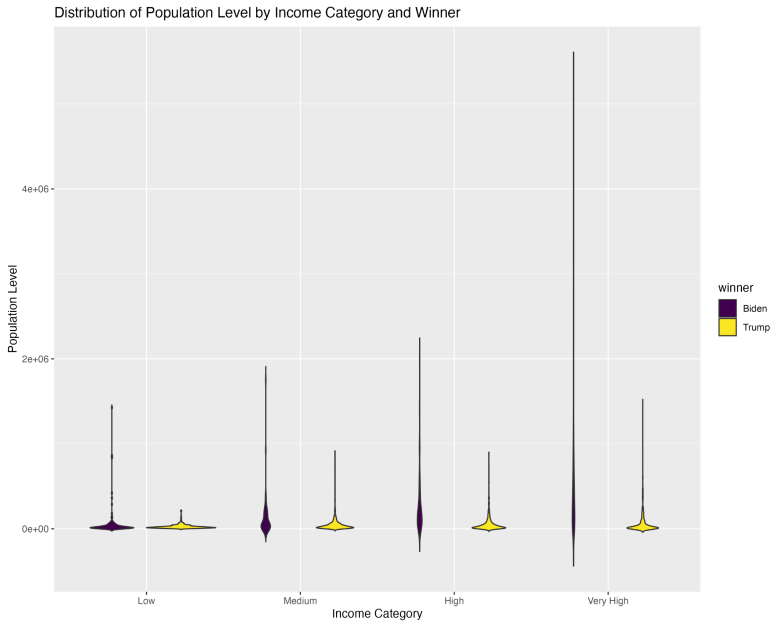


Figure 2.4

The violin plot (Figure 2.4) illustrates the distribution of population levels across income categories, categorized by election winner. Biden-winning counties show a broader range of population levels, especially in higher income categories, compared to Trump-winning counties. This suggests that higher income counties with greater population diversity are more likely to vote for Biden.

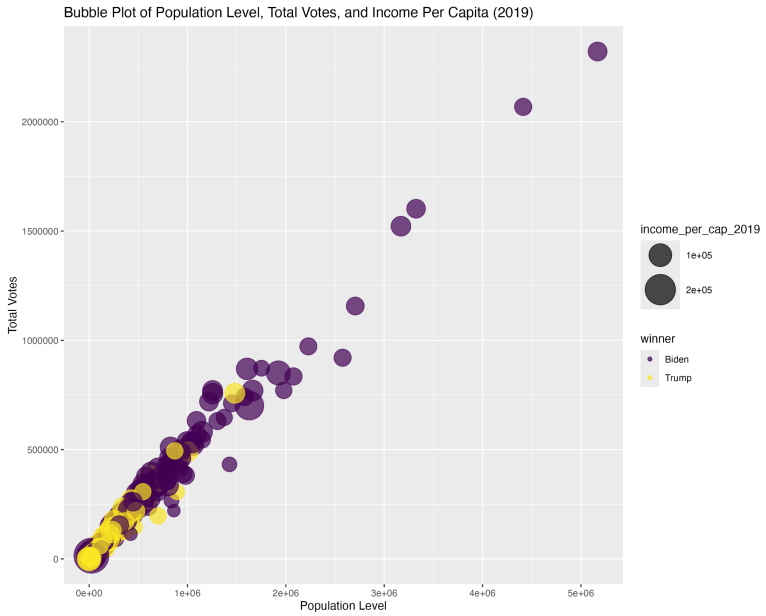


Figure 2.5

The bubble plot (Figure 2.5) displays the relationship between population level, total votes, and income per capita (2019), categorized by the election winner. Each bubble represents a county, with size indicating income per capita and color indicating the election winner (Biden or Trump). Biden-winning counties (purple bubbles) tend to have larger bubbles, indicating higher income per capita, and are more dispersed across higher population levels and total votes. Trump-winning counties (yellow bubbles) are generally smaller and more clustered at lower population levels and total votes. This suggests that counties with higher population levels and incomes tend to have more total votes and are more likely to vote for Biden.

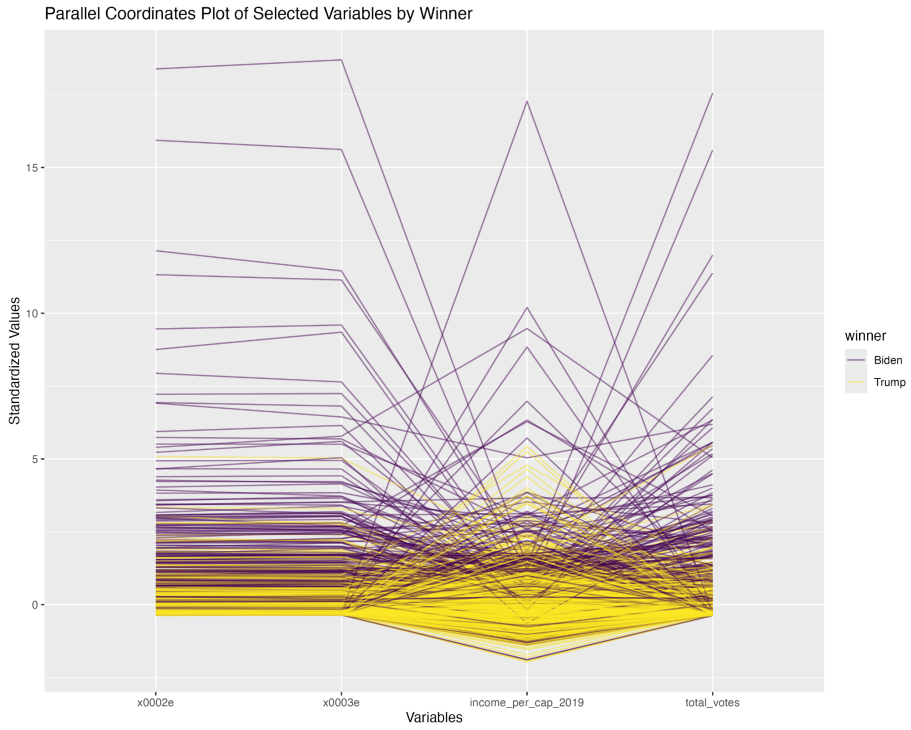


Figure 2.6

Figure 2.6 shows how different variables compare across Biden and Trump-winning counties. Each line represents a county, with the color indicating the election winner (purple for Biden and yellow for Trump). The plot highlights variations and similarities in population levels, income per capita, and total votes between the two groups. Biden-winning counties generally exhibit higher population levels and income per capita compared to Trump-winning counties.

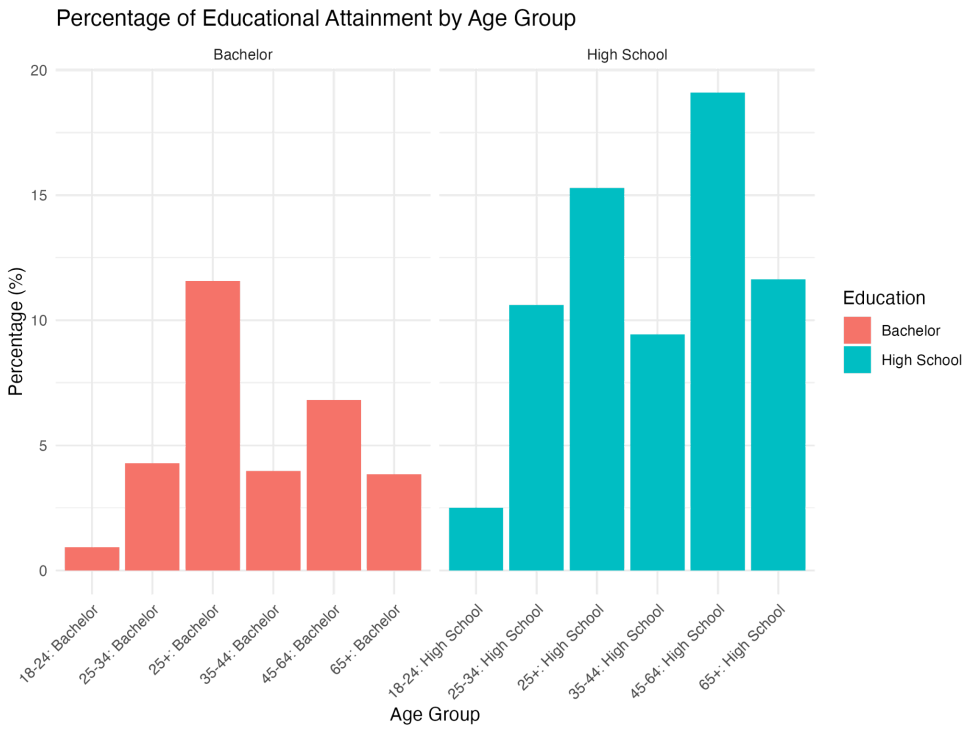


Figure 2.7

Figure 2.7 shows the percentage of high school graduates and bachelor's degree holders across different age groups. The chart is divided into two parts: the left side shows the percentage of bachelor's degree holders, and the right side shows the percentage of high school graduates. Key observations include:

- Age 18-24: The percentage of high school graduates and bachelor's degree holders is relatively low, likely because many individuals are still pursuing higher education.
- Age 25+: The percentage of high school graduates increases significantly, indicating that most individuals in this age group have completed high school education.
- Age 25-34 and 35-44: The percentage of bachelor's degree holders peaks in these age groups, suggesting that more individuals have completed university education.
- Age 45-64 and 65+: The percentage of high school graduates rises again, while the percentage of bachelor's degree holders remains relatively stable, reflecting a lower proportion of higher education attainment among the older population.

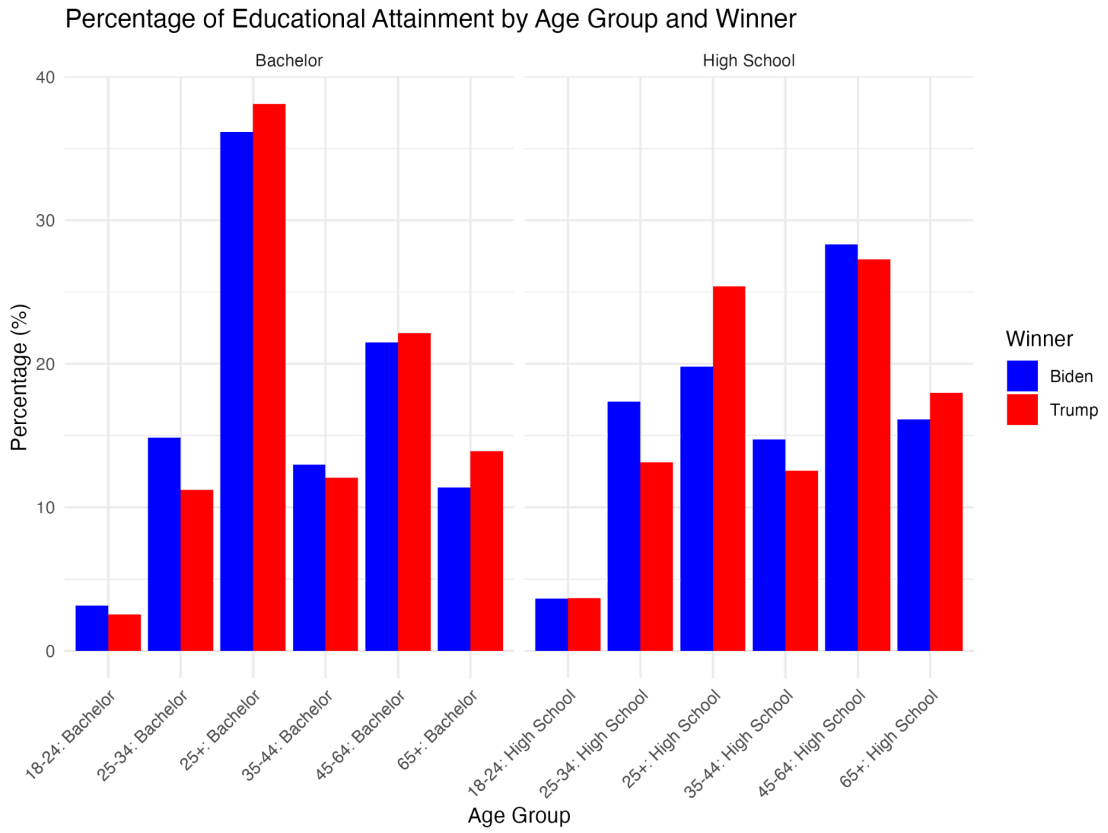


Figure 2.8

Figure 2.8, building on Figure 2.7, shows the percentage of high school graduates and bachelor's degree holders among Biden and Trump supporters across different age groups. The chart is divided into two parts: the left side shows the percentage of bachelor's degree holders, and the right side shows the percentage of high school graduates. Key observations include:

- Biden Supporters: The percentage of bachelor's degree holders is higher, especially in the 25-34 and 35-44 age groups.
- Trump Supporters: The percentage of high school graduates is higher, particularly in the 45-64 and 65+ age groups.
- Comparison: Biden supporters have a higher proportion of bachelor's degree holders in younger age groups, while Trump supporters have a higher proportion of high school graduates in older age groups.

3 Preparation

3.1 Preprocessing/Recipes

First, we remove id and name because the variable of id is not important and not considered as a predictor variable here and the variable of name about information of state and county is not included in the test set. Then, we transform the character variable winner into factor; also, we transform test id into factor so that it will not be log transformed or normalized. After these basic steps, we apply different recipes for various models. For the Logistic Regression, K Nearest Neighbor (KNN), and decision tree, we apply 3 different recipes on each model. For Random forest and Boosting Tree, we use only the base recipe.

3.1.1 Base Recipes

First, we transform the variable "x2013_code," which represents the urban/rural codes, into factors. After examining the correlation matrix, we discover high collinearity between columns. To address this issue, we convert raw counts into proportions. For example, "Total population: Male" becomes the proportion of males in the total population after division. Thus, we divide every column, except for the response variable, area code, and any other possible nominal variables (though there are none in this case), by the total population to get the proportions. Next, we normalize the population variable to eliminate the scale effect. We then check for any infinite values resulting from the transformations and convert these infinite values to NA. Finally, for both the NA values from infinite transformations and missing GDP values for the country from 2016 to 2020, we impute them with the mean values of the corresponding variables.

3.1.2 Filter Recipe

Based on the base recipe, We update it by adding step_corr() for all numeric predictor variables with threshold in the tuning situation. That is, we will remove variables of high correlation.

3.1.3 PCA Recipe

Based on the base recipe, we update it by adding step_pca() for all numeric predictor variables with number to components to remain as new predictor variables in the tuning situation. Therefore, we reduce 128 variables to less than 10 variables with most variance and remain most of the original information as well.

3.1.4 Recipe for Boosting Tree

The recipe for boosting trees has only 1 part that's different from the base recipe. We apply one hot-encoding with step_dummy() on the only categorical variable, "x2013_code", which represents the urban/rural codes.

3.1.5 Recipe for Stacks()

The recipe for the ensemble of models created by stack() uses exactly the same recipes created for KNN, random forest, and boosting tree models.

4 Model Analysis

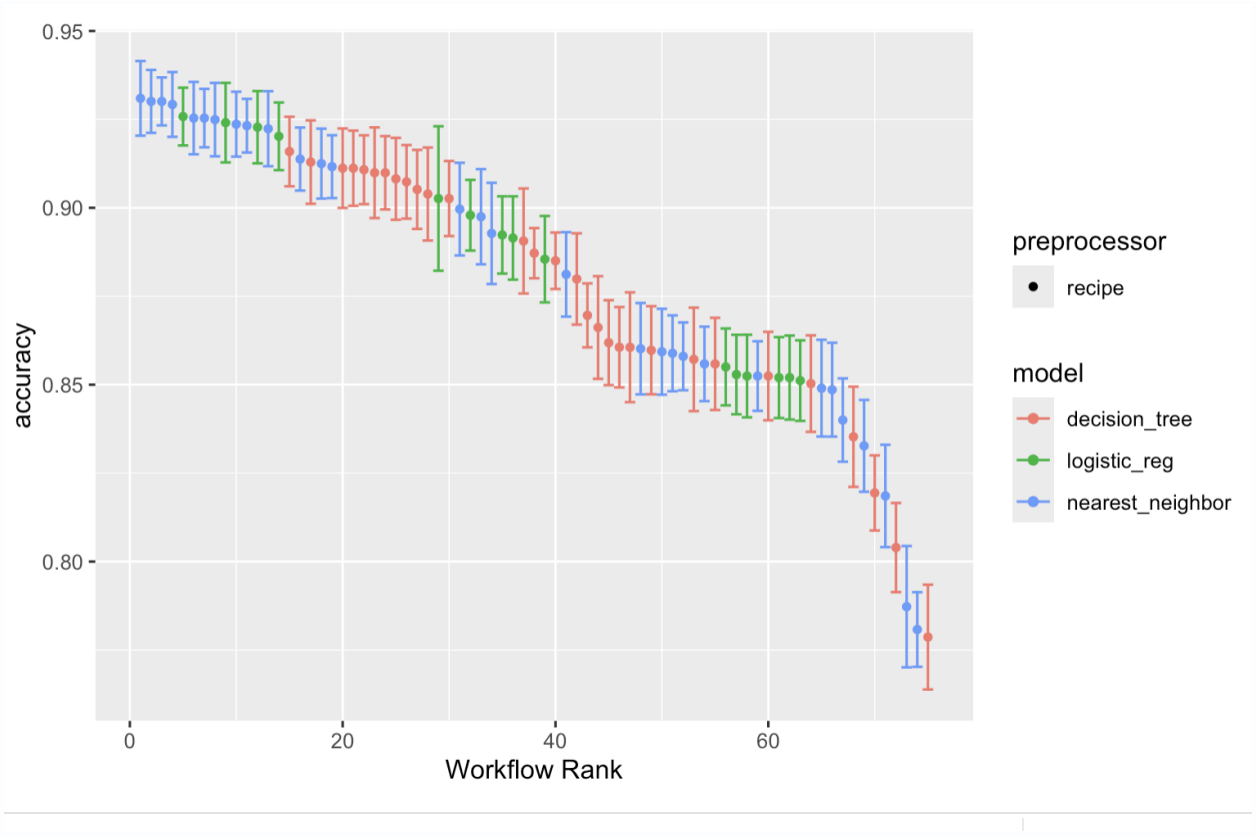
4.1 Candidate Models

Throughout the process of improving the accuracy score of the data, we used multiple types of model, including logistic regression model, KNN model, decision tree model, random forest model, boosting tree model, and an ensemble of models created by using KNN, random forest, and boosting tree. For most models we tuned some parameters inside the model and tried to get a highest score on accuracy, and the list of our settings is created in the table under this paragraph.

Model Iden	Model Type	Engine	Recipe	Hyperparameters
knn_model	KNN	kknn	3.1.1, 3.1.2, 3.1.3	neighbors, weight_func
logistic_result	Logistic Regression	glm	3.1.1, 3.1.2, 3.1.3	N/A
dt_model	Decision Tree	rpart	3.1.1, 3.1.2, 3.1.3	cost_complexity, min_n
boost_tree_model	Boosting Tree	xgboost	3.1.4	trees, min_n(), learnrate, stop_iter, tree_depth
rf_model	Random Forest	ranger	3.1.1	trees, min_n
vote_data_st	Ensemble using KNN, RF, and Boosting Tree	kknn, xgboost, ranger	3.1.1, 3.1.2, 3.1.3, 3.1.4	trees, min_n, neighbors, weight_func

4.1.1 Logistic Regression, KNN, Decision Tree

For the first 3 simpler models (logistic regression, KNN, decision tree), we set up a workflow set that combines each of the preprocessing recipes with each classification model, allowing for a cross-product of preprocessing methods and models. Then, we define a 10-fold cross-validation scheme on the training data. After these steps, we use workflow_map and tune_grid to perform hyperparameter tuning across the different workflows using grid search, evaluating model performance based on accuracy.



rank <int>	mean <dbl>	model <chr>	wflow_id <chr>	.config <chr>
1	0.9309471	nearest_neighbor	simple_knn	Preprocessor1_Model06
2	0.9257969	logistic_reg	filter_logit	Preprocessor05_Model1
3	0.9223653	nearest_neighbor	filter_knn	Preprocessor03_Model1
4	0.9159165	decision_tree	simple_dt	Preprocessor1_Model07
5	0.9129324	decision_tree	filter_dt	Preprocessor03_Model1
6	0.9026375	logistic_reg	simple_logit	Preprocessor1_Model1
7	0.8618796	decision_tree	pca_dt	Preprocessor1_Model4
8	0.8601702	nearest_neighbor	pca_knn	Preprocessor2_Model3
9	0.8524431	logistic_reg	pca_logit	Preprocessor4_Model1

We notice that simple_knn has the highest mean, which means KNN with the base recipe performs the best on accuracy score, so we figure out the hyperparameters of model6 in simple KNN and set neighbors = 14, weight_func = "optimal".

neighbors <int>	weight_func <chr>	.metric <chr>	.estimator <chr>	.estimate <dbl>	.config <chr>
11	biweight	accuracy	binary	0.9098712	Preprocessor1_Model01
2	cos	accuracy	binary	0.9141631	Preprocessor1_Model02
6	epanechnikov	accuracy	binary	0.9055794	Preprocessor1_Model03
4	gaussian	accuracy	binary	0.9313305	Preprocessor1_Model04
5	inv	accuracy	binary	0.9141631	Preprocessor1_Model05
14	optimal	accuracy	binary	0.9055794	Preprocessor1_Model06
8	rank	accuracy	binary	0.9184549	Preprocessor1_Model07
8	rectangular	accuracy	binary	0.9012876	Preprocessor1_Model08
10	triangular	accuracy	binary	0.9141631	Preprocessor1_Model09
13	triweight	accuracy	binary	0.9055794	Preprocessor1_Model10

Figure: simple KNN performance of 1 Folder over total 10 Folder

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
accuracy	binary	0.93094714	10	0.006414017

Figure: Accuracy result for simple KNN

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
accuracy	binary	0.9257969	10	0.004965388

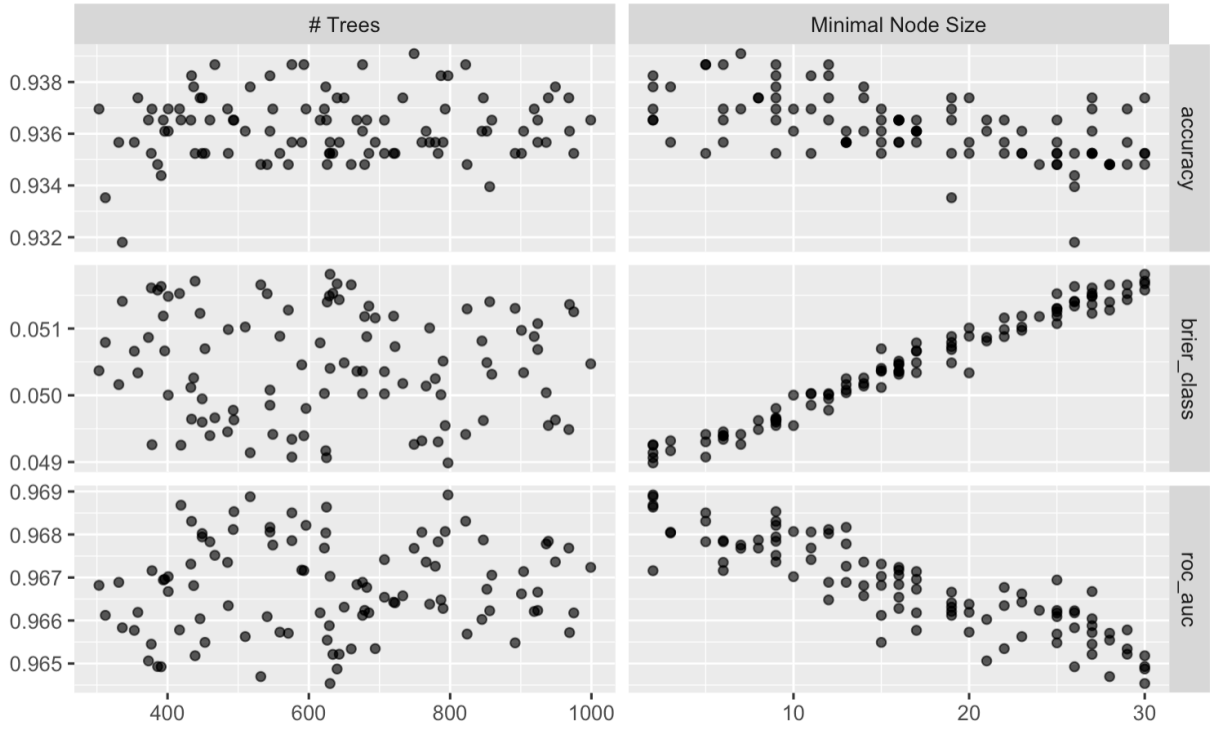
Figure: Accuracy result for filter logistic regression

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
accuracy	binary	0.91591651	10	0.005973366

Figure: Accuracy result for simple decision tree

4.1.2 Random Forest

After these simpler models, we try to use more complicated models. For Random forest, we tune hyperparameters trees and minimum nodes with the range 2 to 30 for minimum nodes and 300 to 1000 for trees. We choose to sample random 100 combinations of trees and min_n. Based on the result, we set trees = 749, min_n = 7.



trees <int>	min_n <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
749	7	accuracy	binary	0.9390961	10	0.005200464	Preprocessor1_Model086
576	5	accuracy	binary	0.9386725	10	0.006298346	Preprocessor1_Model070
593	6	accuracy	binary	0.9386725	10	0.006133719	Preprocessor1_Model082
822	5	accuracy	binary	0.9386706	10	0.006105220	Preprocessor1_Model021
467	9	accuracy	binary	0.9386706	10	0.005795664	Preprocessor1_Model053

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.9373867	10	0.006799228	Preprocessor1_Model1

4.1.3 Boosting Tree

For Boosting Tree, we tune hyperparameters trees and minimum nodes with the range 2 to 30 for minimum nodes and 300 to 1000 for trees. So we set trees = 493, min_n = 2 based on the ranking of accuracy. Then, we add stop_iter = 5, tree_depth = 8 to prevent overfitting. Finally, we tune learning rates with range 0.01 to 0.1. We get a learning rate = 0.06 here. In the whole process, we also apply tune_bayes() which focus more on areas of the search space that are likely to contain optimal value.

trees <int>	min_n <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>	.iter <int>
493	2	accuracy	binary	0.9468141	10	0.004135042	Iter24	24
504	2	accuracy	binary	0.9463849	10	0.004137896	Iter31	31
483	2	accuracy	binary	0.9459576	10	0.004417952	Iter34	34
447	2	accuracy	binary	0.9459558	10	0.004184997	Iter18	18
446	2	accuracy	binary	0.9459558	10	0.004184997	Iter19	19

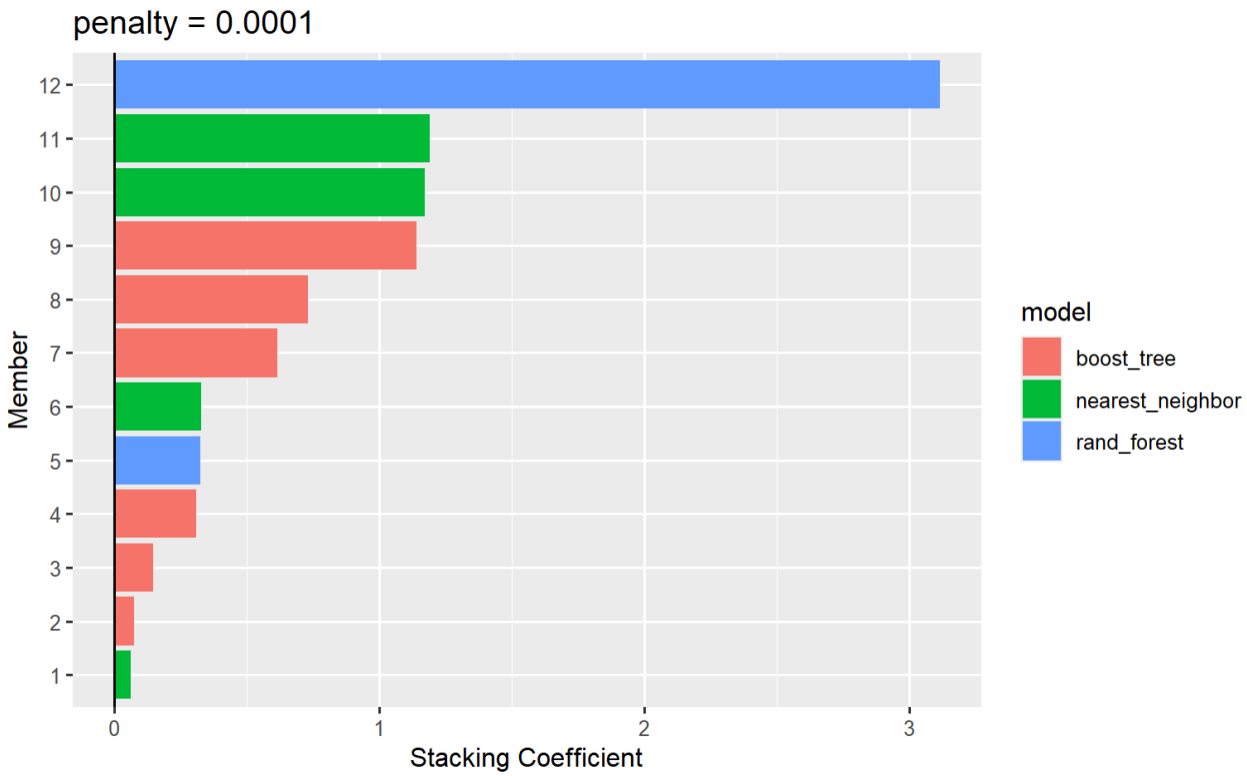
learn_rate <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>	.iter <int>
0.06002660	accuracy	binary	0.9468178	10	0.004548983	Preprocessor1_Model03	0
0.06002707	accuracy	binary	0.9463886	10	0.004551617	Iter3	3
0.08169132	accuracy	binary	0.9459613	10	0.004806943	Iter10	10
0.06538727	accuracy	binary	0.9455284	10	0.004593207	Iter4	4
0.09775985	accuracy	binary	0.9446700	10	0.004241848	Preprocessor1_Model01	0

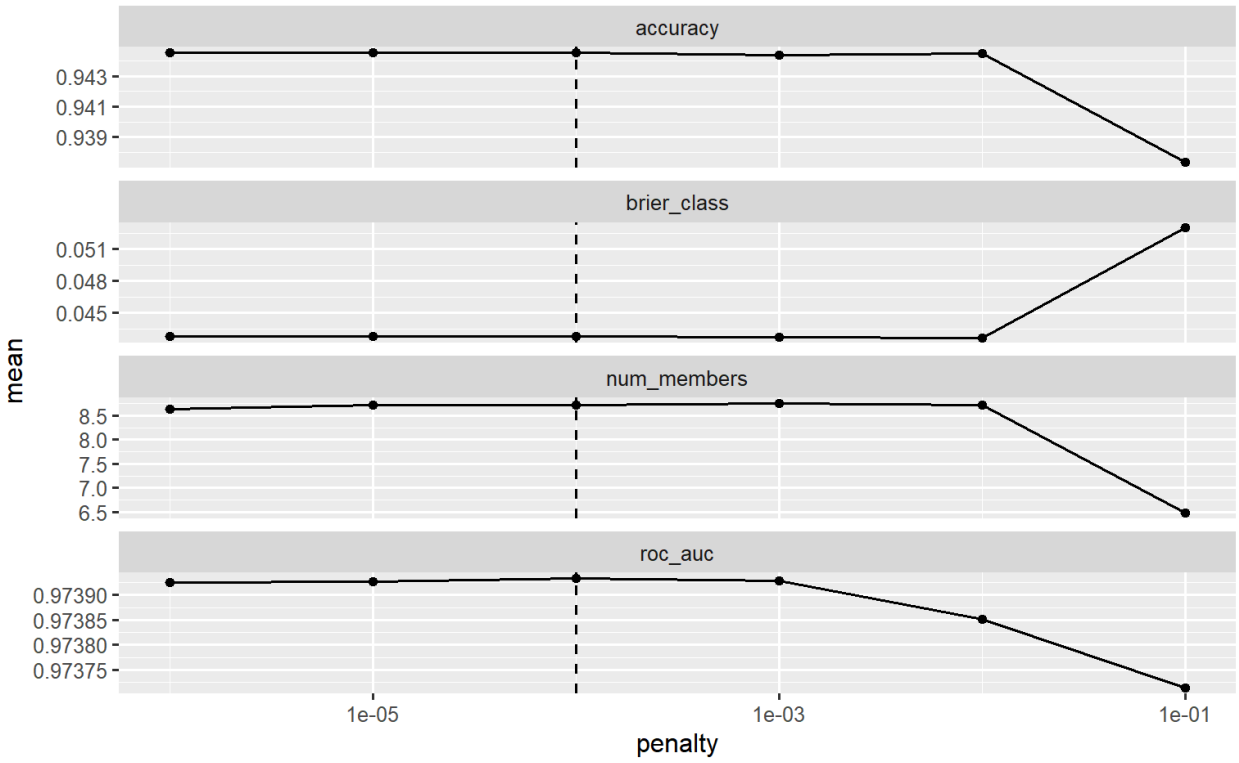
.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.944239	10	0.004077532	Preprocessor1_Model1

1 row

4.1.4 Using Stacks to Create Ensembles of Models

Lastly, to try to further improve the results, we decided to use stacks to create ensembles of models to combine the results of our current models. Our idea of how to create this ensemble is to combine only model types that give us the highest scores in the public leaderboard, therefore we chose the KNN model, the random forest model, and the boosting tree model. Since the random forest model and boosting tree model were giving us the highest result, we decided to set the grid size to 20, while the KNN model will have a grid size of 10. The result graph is shown under this.


penalty = 0.0001
Member
12
11
10
9
8
7
6
5
4
3
2
1
0
1
2
3
Stacking Coefficient
model
boost_tree
nearest_neighbor
rand_forest



penalty <dbl>	mixture <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
1e-06	1	accuracy	binary	0.94548557	25	0.0011158648
1e-06	1	brier_class	binary	0.04230134	25	0.0008968405
1e-06	1	roc_auc	binary	0.97360990	25	0.0009436446
1e-06	1	num_members	Poisson	8.04000000	25	0.5670978752

After fitting our combination of the models, we got a result of:

Submission and Description	Private Score	Public Score
stack2.csv Complete (after deadline) · TIANLIN YUE · 17h ago	**0.92975**	**0.95815**

This result is high enough, but it is not higher than our result from Random Forest and Boosting Tree, therefore we decide to go back and use those 2 models as our final result. Since the Random Forest Model gives slightly better results, we choose it as our final model.

4.2 Model Evaluation

The result scores our candidate models are shown below:

1. Logistic Regression

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
accuracy	binary	0.9257969	10	0.004965388

2. KNN Model

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
accuracy	binary	0.93094714	10	0.006414017

3. Decision Tree

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
accuracy	binary	0.91591651	10	0.005973366

4. Boosting Tree

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.944239	10	0.004077532	Preprocessor1_Model1

1 row

5. Ensemble Using Stacks()

penalty <dbl>	mixture <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
1e-06	1	accuracy	binary	0.94548557	25	0.0011158648
1e-06	1	brier_class	binary	0.04230134	25	0.0008968405
1e-06	1	roc_auc	binary	0.97360990	25	0.0009436446
1e-06	1	num_members	Poisson	8.04000000	25	0.5670978752

6. Random Forest(Final Model)

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.9373867	10	0.006799228	Preprocessor1_Model1

Model Identifier	Recipe	Metric score (accuracy)	SE
knn_model	base	0.93094	0.00064
logistic_result	filter	0.92579	0.00496
dt_model	base	0.91591	0.00597
boost_tree_model	base with dummy	0.94423	0.00407
rf_model	base	0.93738	0.00679
vote_data_st	base	0.94548	0.00111

5. Discussion

5.1 Selection of the final model

In this project, we considered and evaluated multiple classification models, including Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree, Random Forest, Boosting Tree models, and model ensembles. Each model was tested under different preprocessing schemes, and the Random Forest model was ultimately selected as the final model for generating predictions. This decision was based on a comprehensive evaluation of model performance, particularly in terms of accuracy and robustness.

Firstly, the Logistic Regression model, despite being simple and interpretable, is limited in its performance when dealing with highly nonlinear data. Similarly, the KNN model, which classifies data points based on their nearest neighbors, showed good performance in some cases but has high computational complexity, especially with large datasets. The Decision Tree model, while capable of capturing nonlinear relationships, tends to overfit the data when using a single tree, leading to poor generalization.

Through 10-fold cross-validation and hyperparameter tuning, we found that KNN had the highest accuracy in certain settings, while Logistic Regression and Decision Tree models performed relatively weaker.

Secondly, the Boosting Tree model, as an ensemble learning method, improves performance by iteratively correcting the errors of previous models. However, Boosting models are prone to overfitting if not carefully tuned. During our tuning process, we found that Boosting models required meticulous adjustment of multiple hyperparameters and were sensitive to noise in the training data. Although Boosting models performed well after tuning, their complexity and sensitivity made them less stable compared to the Random Forest model in this study.

Finally, the Random Forest model, by aggregating multiple decision trees, significantly enhanced classification performance. It effectively handled high-dimensional data and complex feature relationships while maintaining strong resistance to overfitting. Through 10-fold cross-validation and hyperparameter tuning, we optimized the number of trees and the minimum number of nodes, ensuring stable and efficient predictive performance under various preprocessing schemes. Compared to other models, Random Forest exhibited superior accuracy, robustness, and capability to handle data complexity. Hence, the Random Forest model was selected as the final model for generating predictions.

5.2 Strengths and Weaknesses

The Random Forest model was chosen as the final model for generating predictions due to its outstanding performance. However, like any model, it has its unique advantages and disadvantages. Analyzing these helps us understand why it was preferred and where it might face limitations.

Firstly, the Random Forest model excels in performance, particularly in handling complex nonlinear relationships. By aggregating multiple decision trees, Random Forest

effectively reduces the risk of overfitting seen in single-tree models, thereby improving generalization. This ensemble approach also allows Random Forest to maintain high accuracy with high-dimensional data. Additionally, Random Forest provides feature importance metrics, offering valuable insights into which variables are most critical for prediction. However, the Random Forest model is computationally intensive. Training numerous decision trees requires substantial computational resources and time, especially with larger datasets. While Random Forest can provide feature importance, its overall model complexity makes it difficult to interpret individual predictions, reducing model interpretability.

On the other hand, the Random Forest model is robust in handling missing values and outliers. Its use of different sample subsets and feature subsets during training provides a degree of resistance to noise and missing data. However, it may not perform as well with high-dimensional sparse data, such as text data, compared to some linear models. In such cases, the advantages of Random Forest might be limited, necessitating the combination with other methods for optimization. Overall, the strengths of the Random Forest model lie in its powerful predictive capabilities and feature importance insights, while its weaknesses are in computational cost and interpretability.

5.3 Possible Improvements

While the Random Forest model performed well in this study, there is still room for improvement. By further optimizing the model and adding additional datasets, we can enhance its performance and predictive accuracy. Some techniques applies to all models we tested, and could help us improve our model ensemble result.

Firstly, further feature engineering could significantly improve the model's performance. Although we have already applied feature normalization and proportionality adjustments, exploring more feature interactions and derived features, such as products or ratios between variables, could reveal hidden patterns and relationships, thereby enhancing predictive capabilities. Additionally, trying different hyperparameter tuning methods, such as Bayesian optimization (tune_bayes()), could

further optimize the Random Forest model's parameters. Bayesian optimization can more effectively explore the parameter space compared to grid search or random search, finding better parameter combinations.

In terms of additional data, we can check external public dataset published by government or non-profit organizations and incorporate into our analysis. To start with, climate and environmental data might correlate with election outcomes. Certain climate conditions on campaign speech and final voting process could influence voter turnout and behavior, indirectly affecting election results. Social media data can also provide valuable supplementary information. Analyzing the discussion intensity and sentiment trends in different regions can offer better insights into voter inclinations. Lastly, crime rate data and public safety-related data may also significantly impact election outcomes. Voters in high-crime areas may have different concerns and priorities, understanding these patterns can help improve the model's predictive power.

For our model ensemble created by stacks function, we should probably add more model types in the ensemble to improve our predictive power. Our initial idea on building this ensemble is using only models that give a superior result in our prior testing to reduce variability and time consumption. However, as a combination of different models, the ensemble probably should include more types to ensure a higher predicted accuracy because different models have strength in interpreting different kinds of data. By including only random forest, boosting tree, and random forest models, the ensemble might compromise every model’s peak predictive ability, but did not improve its power on predicting values that these models cannot accurately predict.

6 Conclusion

In this report, we focus on predicting the outcome of 2020 US Presidential Election at the county level using the dataset of 3,111 counties. Throughout the data analysis and model evaluation, many machine learning techniques were used, including logistic regression, KNN, decision trees, random forests, and boosting tree models. And the primary focus of the project was to identify the most accurate

predictive model for determining whether Biden or Trump would win each county based on the demographic and education level. Moreover, during the models testing, random forest models seem to be most effective with highest accuracy. Which shows that it has strong predictive performance and interpretability for analyzing election data. These findings highlight the importance of using advanced machine learning techniques for political analysis.

7 Appendix

7.1 Final Annotated Script

```
library(tidyverse)
library(tidymodels)
library(ggplot2)
# load training data
train <- read_csv("train_class.csv")
# convert winner, area code into factor
train$winner <- as.factor(train$winner)
train$x2013_code <- as.factor(train$x2013_code)
train <- train %>% select(-id, -name)
# remove id and state and county name

# load testing data
test <- read_csv("test_class.csv")
# convert id, area code into factor
test$id <- as.factor(test$id)
test$x2013_code <- as.factor(test$x2013_code)

set.seed(123)
train_folds <- vfold_cv(train, v = 10)

# creating basic recipe
base_recipe <- recipe(winner ~ ., data = train) %>%
  step_mutate_at(all_numeric_predictors(), -matches("x0001e"),
-matches("total_votes"), -all_nominal_predictors(), fn = ~ . /
x0001e) %>%
```

```
  step_normalize(all_of("x0001e"), all_of("total_votes")) %>%
# else, without scaling, too large, diverge
  step_mutate_at(all_numeric_predictors(), fn = ~
ifelse(is.infinite(.), NA, .)) %>% # impute missing values in
numeric predictors with the mean
  step_impute_mean(all_numeric_predictors())

# tune trees and minimum nodes in random_forest
rf_model <- rand_forest(trees = tune(), min_n = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")

# create workflow, add model, recipe
rf_wkfl <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(base_recipe)

# extract the parameter set and update the range for trees and
min_n
recipes_param <- extract_parameter_set_dials(rf_model)
recipes_param <- recipes_param %>% update("trees" =
deg_free(c(300, 1000)),
                                          "min_n" =
deg_free(c(2, 30)))

# tune the random forest model using grid
winner_tune<- rf_wkfl %>%
  tune_grid(
    train_folds,
    grid = recipes_param %>% grid_random(size = 100))

# plot the result
autoplot(winner_tune)
show_best(winner_tune, metric = "accuracy")

workflow_fit <- bt_wkfl %>% fit(train) # fit training data
pred <- predict(workflow_fit, new_data = test) # prepare winner
for test data
# prepare submission file with id and predicted winner
```

```
predictions <- test %>% select(id) %>%
  bind_cols(pred) %>%
  rename(winner = .pred_class)
predictions %>%  head(15)
write_csv(predictions, "RF_Predictions.csv") # save predictions
to csv file
```

7.2 Team member Contributions

Tianlin Yue: Recipe and Model parts specifically for Stacks(), Stacks part in discussion

Rueiyu Chang: Finish the introduction and conclusion sections, and overall report coordination.

Kai Liang: Discussion section

Yiming Xue: Recipe and Model parts for all other models.

Ziwei Guo: Data Visualization and Data Analysis

8 Citation

Igielnik, Ruth, et al. "Behind Biden's 2020 Victory." *Pew Research Center*, Pew Research Center, 30 June 2021, www.pewresearch.org/politics/2021/06/30/behind-bidens-2020-victory/.