ST310 Final Project

A comparison of predictive machine learning models for Asian American societal satisfaction.

37652, 43259, 38755

Tue/03/Jun

Contents

Introduction

Motivating question

Our motivating question for training predictive models on our selected dataset is: **What factors best predict Asian American attitudes towards the way things are going in the United States?** More operationally, how can we best predict whether Asian Americans are either satisfied or dissatisfied.

Literature Review

Predicting whether respondents are satisfied with the direction of the United States is a vital endeavor in the realm of public opinion research. Our focus on Asian American communities, whose perspectives have historically been underrepresented, is particularly significant. Employing machine learning techniques to forecast such sentiments can enhance our understanding of societal dynamics, inform policy decisions, and promote inclusive governance. This could include predicting the outcome of electoral results in swing states where the vote depends on Asian Americans. In swing states, Asian American voters have shown significant electoral influence. Analysis by AAPI Data (2024) highlights the potential impact of Asian American, Native Hawaiian, and Pacific Islander (AANHPI) voters in swing states. It emphasises that in states like Nevada, Georgia, and Pennsylvania, the AANHPI electorate could play a decisive role in election outcomes due to their growing numbers and increasing political engagement. For example, in the 2024 presidential election, Asian American voters played a crucial role in swing states (MyAsianVoice, 2024). Although these voters represent a smaller percentage of the electorate in these states, their support can be decisive in close elections.

Public satisfaction serves as a barometer for the nation's political, economic, and social health. Gallup's longitudinal data reveals fluctuations in national satisfaction, with notable declines during periods of political unrest or economic downturns. Understanding these trends is crucial for policymakers to address public concerns effectively. More interpretable models can be useful in highlighting the most important factors that influence satisfaction, which can be used to inform policies to improve satisfaction of citizens overall.

For Asian Americans, satisfaction levels can reflect unique experiences shaped by immigration, cultural integration, and socio-political challenges. A 2023 poll by AAPI Data and The Associated Press-NORC Center for Public Affairs Research indicated that approximately 70% of Asian Americans and Pacific Islanders believe the country is headed in the wrong direction, highlighting the need for targeted policy interventions (Brown and Sanders, 2023).

The advent of ML has revolutionised the analysis of public opinion. ML models can process vast datasets to identify patterns and predict outcomes with high accuracy. For instance, Khan et al. (2024) demonstrated the potential of ML algorithms to predict life satisfaction with an accuracy of 93.80% using clinical and biomedical large language models (LLMs). Our analysis will employ techniques more similar to Oparina, Kaiser, Gentile, et al. (2025), who explored the applications of tree-based algorithms and penalised regression to the prediction of wellbeing using survey data similar to ours. They found that tree-based approaches consistently perform better than conventional linear models. We will test both of these techniques in our analysis.. The application of feed-forward neural networks was also evaluated in their preliminary analysis, but this was not fruitful enough for further consideration.

Asian Americans represent a diverse and rapidly growing demographic in the U.S. However, capturing their perspectives poses challenges due to linguistic diversity, cultural nuances, historical underrepresentation in surveys and relatively low population. Efforts like the Pew Research Center's comprehensive surveys aim to bridge this gap by providing more accurate and representative data. Our data comes from a national cross-sectional survey conducted for Pew Research Center by Westat. The survey reached a nationally representative group of respondents, either online or by mail, resulting in 7,006 interviews with Asian American adults. The use of online surveys is one way to overcome the challenges of reporting on Asian American news, "already leading to a small uptick in the publication of Asian American estimates" (Kennedy and Ruiz, 2020).

Given this context, developing models to predict Asian American satisfaction with national direction can provide insights into their voting behavior, which is particularly valuable in swing states where their votes

can influence the outcome of elections. Predicting public satisfaction is essential for fostering an inclusive and responsive society. Integrating machine learning tools with culturally competent research practices can ensure that the shaping of future policy is representatively informed and attuned to key voters.

Data

We used data from the 2022-23 Pew Research Center's Survey of Asian Americans, which asked a nationally representative sample of Asian American adults about their experiences living in, and views of, the United States. The data is primarily comprised of categorical predictors, and our outcome variable is binary. The outcome is the response to the question: "Are you satisfied or dissatisfied with the way things are going in this country today?", with the responses of either "Satisfied" or "Dissatisfied".

We cleaned and preprocessed the data to address missing values, and sparsity. After cleaning, we had 51 predictors overall and 2,653 observations.

IID Assumption

While no survey dataset is perfectly independent and identically distributed (IID), the design and execution of the Pew Research Center's survey support the practical approximation of this assumption for modeling purposes. The dataset is drawn from a cross-sectional, nationally representative sample of Asian American adults, selected using stratified address-based sampling supplemented by surname list frames to ensure adequate subgroup representation. All respondents completed the same extended questionnaire under standardised, self-administered conditions, reducing measurement variation. The use of post-stratification weights further aligns the sample distribution with the target population. Although the sampling design is complex and some census tracts were excluded, the absence of panel data, interviewer influence, or known clustering effects means responses can be reasonably treated as independent. Therefore, it is appropriate to assume the data are approximately IID after accounting for survey weights.

Meeting requirements

The 5 requirements of the project have been met by our models. At the top of each section is a brief statement of which requirement is met by the model and why. Here is a summary of which models meet each requirement:

1. Logistic regression
2. Gradient Descent
3. Single tree
4. Penalised Logistic Regression (Lasso)
5. Random forest and gradient descent

Analysis

```
data <- read.csv("data/2022-23 Survey of Asian Americans_Public use file.csv")

library(dplyr)        # Data manipulation and transformation (part of the tidyverse)
library(tree)         # Building and visualising decision tree models
library(tidyr)        # Tidying data
library(ggplot2)      # Creating static visualizations (part of the tidyverse)
library(glmnet)       # Penalised regression models (we used for Lasso)
library(tidyverse)    # Meta-package including ggplot2, dplyr, tidyr, readr, etc.
library(randomForest) # Building random forest models (ensemble learning)
library(knitr)        # Dynamic report generation (since we are working in R Markdown)
```

```r
library(kableExtra)   # Enhancing 'knitr::kable' tables for the report

set.seed(123)
```

```r
# Create an empty dataframe to store all the misclassification rates of all our models
df_results <- data.frame(
  Model = character(),
  Misclassification_rate = numeric(),
  stringsAsFactors = FALSE
)
```

Data Preprocessing

```r
data_final <- data %>%
  # Remove metadata/unnecessary columns
  select(-PARTICIPANTCODE_PUF, -PARTICIPANTCODE_RUF, -SCREENERSTARTDATE,
         -EXTENDEDSTARTDATE, -EXTENDEDCOMPLETEDATE, -EXTENDEDMODE,
         -EXTENDEDSUBMITLANGUAGE, -FRAME, -SCREENERCOMPLETEDATE, -TRACK,
         -SCREENERMODE, -SCREENERSUBMITLANGUAGE) %>% select(1:68) %>%

  # Handling missing values
  mutate(across(where(is.numeric), ~na_if(., 99))) %>% # Replace missing values with NA
  select(where(~ mean(is.na(.)) <= 0.1)) %>% # Keep columns with <=10% missing values
  drop_na() %>% # Drop rows with any remaining NAs

  # Recode variables
  mutate( # Recode binary variables
    SATIS = ifelse(SATIS == 2, 0, SATIS)
  )

# Convert all columns to factors (necessary for classification modelling)
data_final <- data_final %>%
  mutate(across(everything(), as.factor))

# Identify and remove low-information variables
low_info_cols <- data_final %>%
  summarise(across(everything(), ~ max(table(.)) / sum(table(.)))) %>%
  pivot_longer(everything(), names_to = "col", values_to = "dominance") %>%
  filter(dominance > 0.95) %>%  # Adjust threshold as needed
  pull(col)

# Final dataset
data_final <- data_final %>%
  select(-all_of(low_info_cols))
```

We preprocessed the dataset by first removing metadata and administrative columns not relevant to analysis. Numeric variables with coded missing values (99) were converted to NA, and we retained only columns with 10% or fewer missing values, then dropped rows with any remaining missing data. A binary variable (SATIS) was recoded for consistency. All variables were then converted to factors to support classification modeling. Finally, we removed low-information categorical variables, defined as those where a single category accounted for more than 95% of values, as they offer minimal predictive value.

Test/Train split

We used the method from ISLR to perform our test/train split, and used the same split to train and test all models.

```
# Testing and training split
train <-sample(1:nrow(data_final), size = floor(0.8 * nrow(data_final)))
test <- setdiff(1:nrow(data_final), train)
SATIS.test <- data_final$SATIS[test]
```

Logistic Regression

This is our baseline model to use as comparison, satisfying requirement number 1.

Our base model is a logistic model (ISLR p172) due to the binary outcome. This is an approbriate baseline sice it sets a strong, interpretable, and fast-to-compute benchmark. If our more complex models don't outperform logistic regression meaningfully, this will indicate that they may not be worth the added complexity.

```
# Fit using all the training data
logistic <- glm(SATIS ~ ., data = data_final[train, ], family = binomial)
```

```
# Predictions using all of the testing data
probs_logistic <- predict(logistic, newdata = data_final[test, ], type = "response")
pred_logistic <- rep(0, nrow(data_final[test,]))
pred_logistic[probs_logistic > 0.5] = 1

# Table
prop.table(table(pred_logistic))
```

```
## pred_logistic
##         0         1
## 0.7834275 0.2165725
```

```
# Confusion matrix
confusion_matrix <- table(pred_logistic, data_final[test, ]$SATIS)

# Calculate the misclassification rate
misclassification_rate  <- 1 - sum(diag(confusion_matrix)) / sum(confusion_matrix)

# Add to results table
df_results <- rbind(df_results, data.frame(
  Model = "Logistic Regression",Misclassification_rate = misclassification_rate))

# Print the misclassification rate
cat("Misclassifcation rate: ", misclassification_rate)
```

```
## Misclassifcation rate:  0.2391714
```

Classification Trees

Next, we fit several classification trees, both single and random forest (ISLR p353). As stated in our Literature Review, these methods have been shown to offer high predictive accuracy for survey data (e.g. Oparina, Kaiser, Gentile, et al., 2025).

Single tree

This model satisfies requirement number 3, since it is relatively interpretable. See Interpretation and Comparison for a brief sub-section including interpretation of the results and comparison to the baseline model on both predictive accuracy and (in)consistency of interpretations.

```
# Fit tree using training data
SATIS.tree <- tree(SATIS ~ ., data_final, subset = train)

# Make predictions on the test set
SATIS.pred <- predict(SATIS.tree, data_final[test, ], type = "class")
```

```
# Confusion Matrix
conf_matrix <- table(SATIS.pred, SATIS.test)
conf_matrix
```

```
##           SATIS.test
## SATIS.pred   0   1
##          0 316  81
##          1  63  71
```

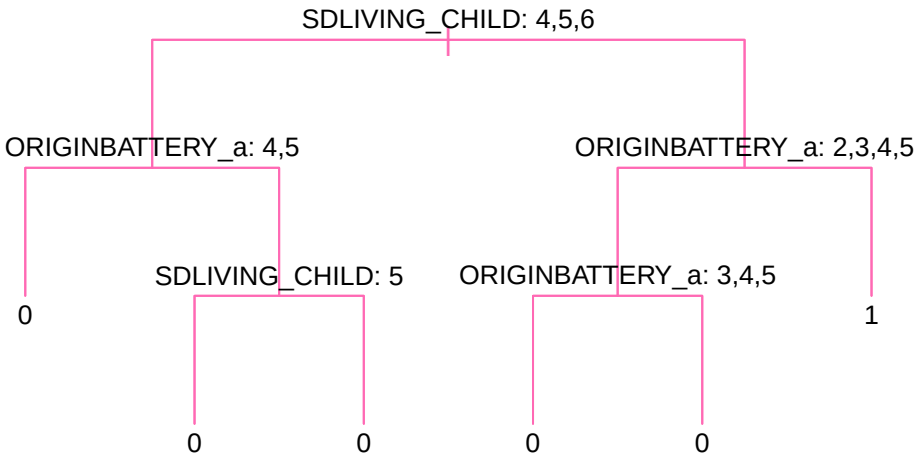
```
# Calculate misclassification rate
misclassification_rate <- mean(SATIS.pred != SATIS.test)

# Add to results table
df_results <- rbind(df_results, data.frame(
  Model = "Single Tree", Misclassification_rate = misclassification_rate))

# Print the misclassification rate
cat("Misclassification Rate: ", misclassification_rate)
```

```
## Misclassification Rate:  0.2711864
```

Overall, the performance is moderate (the misclassification rate is low).

```
# Plot the tree
plot(SATIS.tree,
     type = "uniform",
     col = "hotpink")

text(SATIS.tree,
     pretty = 0,
     cex = 0.8,
     col = "black")
```

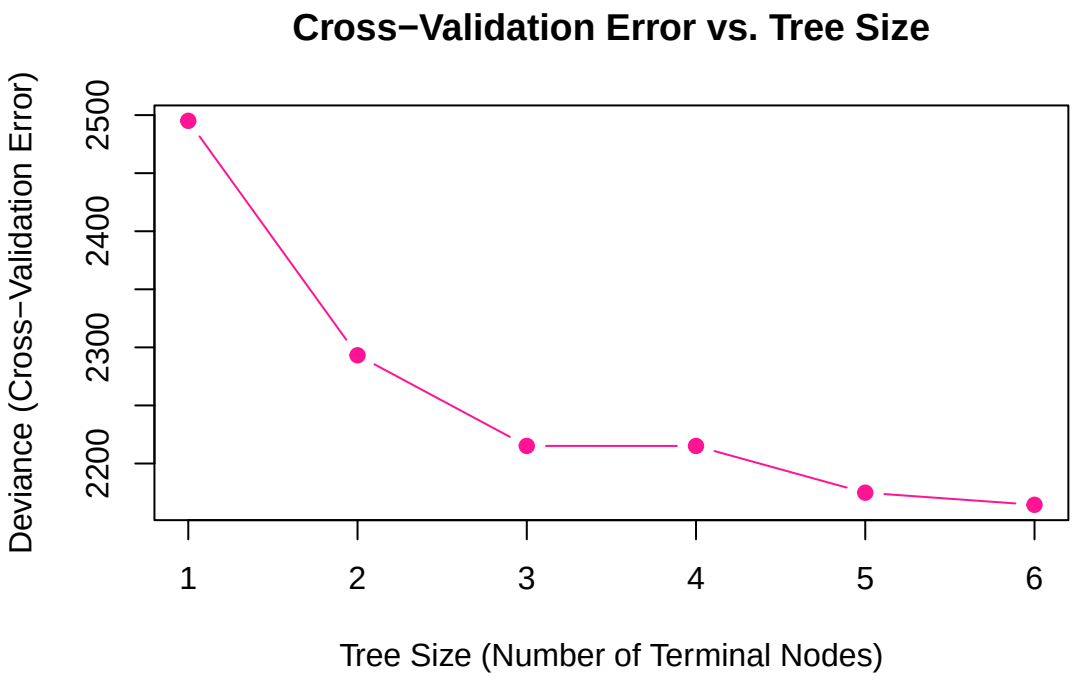
Visualisation



Since multiple splits lead to identical outcomes, pruning the tree is appropriate to remove unnecessary splits that offer not information.

```
# Cross validation
cv.SATIS.tree <- cv.tree(SATIS.tree)
plot(cv.SATIS.tree$size, cv.SATIS.tree$dev, type = "b", pch = 19, col = "deeppink",
     xlab = "Tree Size (Number of Terminal Nodes)",
     ylab = "Deviance (Cross-Validation Error)",
     main = "Cross-Validation Error vs. Tree Size")
```

Cross-validation and Pruning



The 'elbow point' from cross-validation seems to be at 3 nodes. So we will prune the tree to 3 nodes.

```
# Prune tree
prune.SATIS.tree <- prune.misclass(SATIS.tree, best = 3)
SATIS.tree.pred <- predict(prune.SATIS.tree, data_final[test, ], type = "class")

# Confusion Matrix
table(SATIS.tree.pred, SATIS.test)
```

```
##                 SATIS.test
## SATIS.tree.pred   0   1
##               0 316  81
##               1  63  71
```

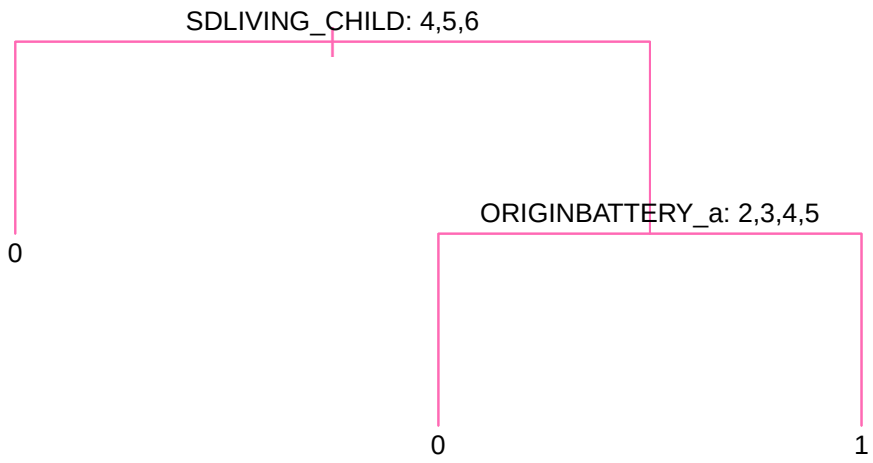
```
# Calculate misclassification rate
misclassification_rate <- mean(SATIS.tree.pred != SATIS.test)

# Add to results table
df_results <- rbind(df_results, data.frame(
  Model = "Pruned Tree", Misclassification_rate = misclassification_rate))

# Print misclassification rate
cat("Misclassification Rate: ", misclassification_rate)
```

```
## Misclassification Rate:  0.2711864
```

The misclassification rate remains the same because no changes to the decisions of the tree have occurred.

```
# Plot the pruned tree
plot(prune.SATIS.tree,
     type = "uniform",
     col = "hotpink")
```

```
text(prune.SATIS.tree,
     pretty = 0,
     cex = 0.8,
     col = "black")
```



Interpretation and Comparison To satisfy requirement number 3.

The classification tree predicts the outcome variable based on only two parameters, namely SDLIVING_CHILD and ORIGINBATTERY_a. The codebook translations are as follows:

SDLIVING_CHILD: When your children are at the age you are now, do you think their standard of living will be…

1. Much better than yours
2. Somewhat better than yours
3. About the same as yours
4. Somewhat worse than yours
5. Much worse than yours
6. Do not have children

ORIGINBATTERY_a: What is your opinion of the U.S.?

1. Very favorable
2. Somewhat favorable
3. Neither favorable nor unfavorable
4. Somewhat unfavorable
5. Very unfavorable

The model predicts general dissatisfaction across the population. However, individuals who are both **optimistic about their children's future** and hold a **favourable opinion of the United States** are more likely to be satisfied with the way things are going in the country today. Conversely, pessimism about

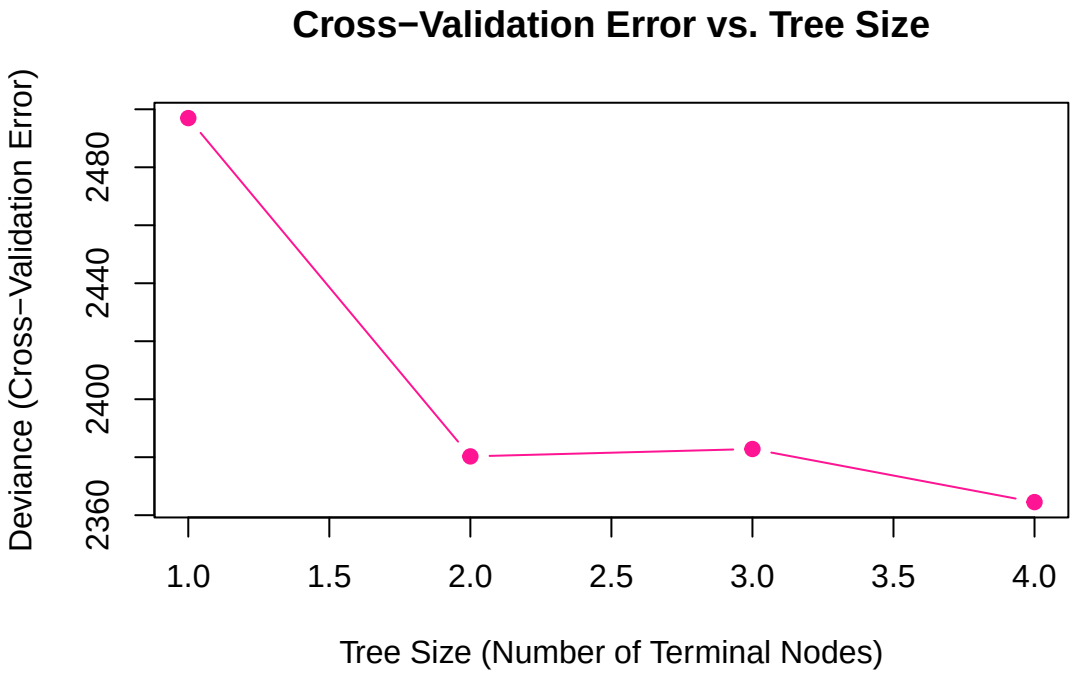
the next generation's future strongly correlates with dissatisfaction. Those who are optimistic about their children's future can feel dissatisfied if they have an unfavourable opinion of the U.S.

Compared to the baseline model, both single classification trees yield higher misclassification rates. This is indicative of the sacrifice of predictive accuracy that has been made for the sake of interpretability for these methods.

This findings do not offer much interesting insight, since these observations merely align with common sense, and would be highly correlated in the minds of most people. Out of curiosity, we will fit another tree after removing the variables SDLIVING_CHILD and ORIGINBATTERY_a from the data, to see if we can learn anything more nuanced about what factors predict satisfaction with the way things are going.

```
# Fit tree on new data
tree2_data <- data_final %>%
  select(-SDLIVING_CHILD, -ORIGINBATTERY_a)

SATIS.tree2 <- tree(SATIS ~ ., data = tree2_data, subset = train)
```

```
# Cross-validation
cv.SATIS.tree2 <- cv.tree(SATIS.tree2)
plot(cv.SATIS.tree2$size, cv.SATIS.tree2$dev, type = "b", pch = 19, col = "deeppink",
     xlab = "Tree Size (Number of Terminal Nodes)",
     ylab = "Deviance (Cross-Validation Error)",
     main = "Cross-Validation Error vs. Tree Size")
```



The 'elbow' point here is 2.

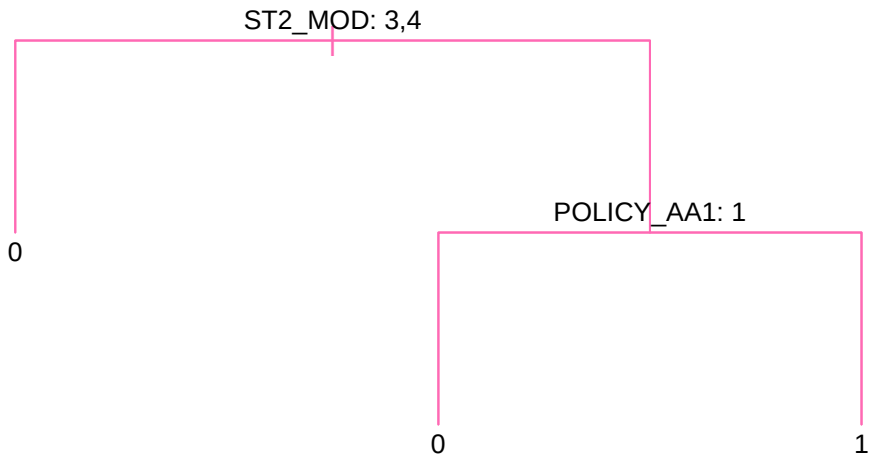
```
# Prune
prune.SATIS.tree2 <- prune.misclass(SATIS.tree2, best = 2)
SATIS.tree.pred2 <- predict(prune.SATIS.tree2, data_final[test, ], type = "class")
table(SATIS.tree.pred2, SATIS.test)
```

```
##                  SATIS.test
## SATIS.tree.pred2   0   1
##                0 349 130
##                1  30  22
```

```r
# Calculate misclassification rate
misclassification_rate <- mean(SATIS.tree.pred2 != SATIS.test)
cat("Misclassification Rate: ", misclassification_rate)
```

```
## Misclassification Rate:  0.3013183
```

```r
# Plot the tree
plot(prune.SATIS.tree2,
     type = "uniform",
     col = "hotpink")

text(prune.SATIS.tree2,
     pretty = 0,
     cex = 0.8,
     col = "black")
```



The next most indicative parameters seem to be

ST2_MOD: Do you think the U.S. immigration system…

1. Does not need to be changed
2. Needs only minor changes
3. Needs major changes
4. Needs to be completely changed

POLICY_AA1: Have you ever heard the phrase “affirmative action”?

1. Yes
2. No

This model predicts general dissatisfaction across the population. However, individuals who believe the **immigration system needs major change** and are **aware of affirmative action** are more likely to be satisfied with the country's direction. Conversely, those less critical of the immigration system tend to be dissatisfied. This is a much more interesting insight as it seems counter to intuitive instinct. One would imagine that those who believe that change is necessary would be more dissatisfied, but the results suggest otherwise. But feeling that the system needs change without awareness of affirmative action (policies and practices designed to address past and ongoing discrimination by promoting opportunities for historically marginalised groups) also leads to dissatisfaction according to the model. This suggests that awareness of affirmative action is key for those who believe change is necessary to feel satisfied. This can be braodly interpreted and generalised to suggest that being unhappy with the way things are (wanting change) does not lead to overall dissatisfaction *as long as* policies are in place to improve things (make the change).

Bagging and Random Forest

The following models satisfy requirement number 5, sacrificing interpretability for predictive accuracy. Bagging and random forest models cannot be visualised so are difficult to interpret, but have been shown to provide high predictive accuracy, particularly for survey/public opinion type data like the Pew data we are working with (e.g. Khan et al., 2024). We will use the `randomForest` package throughout this section.

Bagging For bagging, we use `mtry=57`, forcing the model to include all of our predictors. Then, we tune the number of trees to optimally balance error rate and computational efficiency, indicated by the 'elbow point' of the error rate plot.

```
# Fit bagged tree
bag.tree <- randomForest(SATIS ~ ., data = data_final,
                         subset = train, mtry = 57, importance = TRUE)
```

```
# Predict on test set
SATIShat.bag <- predict(bag.tree, newdata = data_final[-train, ])

# Confusion matrix
table(SATIShat.bag, SATIS.test)
```

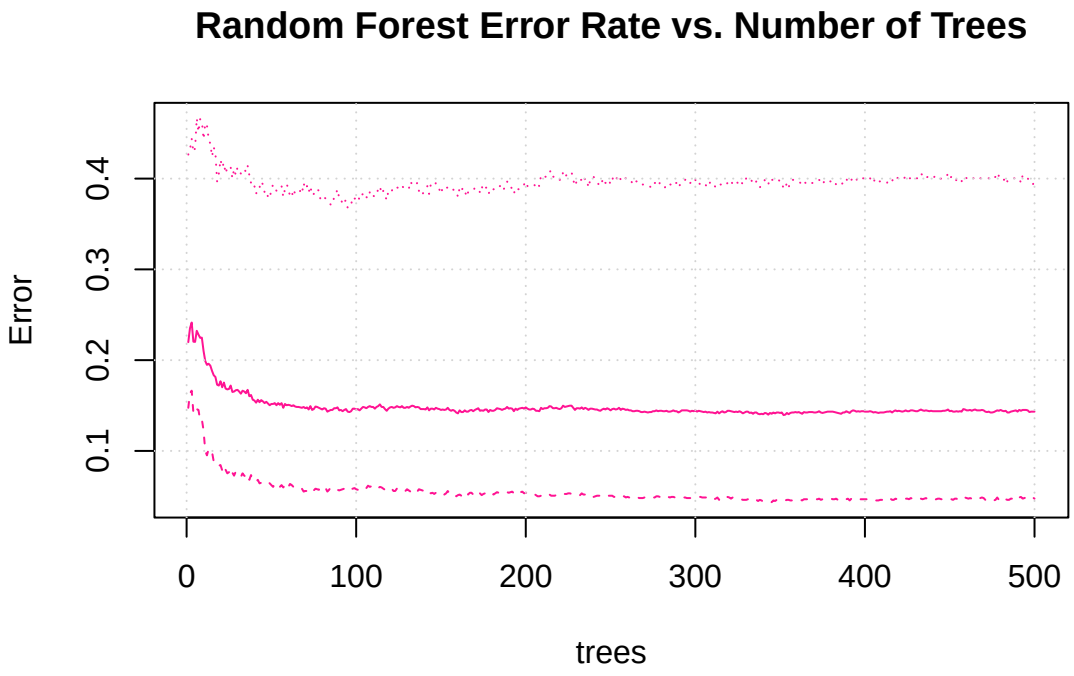
```
##             SATIS.test
## SATIShat.bag   0   1
##            0 355  68
##            1  24  84
```

```
# Calculate misclassification rate
misclassification_rate <- mean(SATIShat.bag != SATIS.test)

# Add to results table
df_results <- rbind(df_results, data.frame(
  Model = "Bagged Tree", Misclassification_rate = misclassification_rate))

# Print misclassification rate
cat("Misclassification Rate: ", misclassification_rate)
```

```
## Misclassification Rate:  0.173258
```

```
# Plot error rate vs num trees
plot(bag.tree,
     main = "Random Forest Error Rate vs. Number of Trees",
```

```
     col = "deeppink")
grid()
```



The above plot illustrates how the predictive accuracy of the random forest model increases as more trees are used. The 'elbow point' of the line is around 70, so I will try bagging with 50 trees next.

```
# Plot forest with 70 trees
bag.tree2 <- randomForest(SATIS ~ ., data = data_final,
                          subset = train, mtry = 57, ntree = 70)
yhat.bag <- predict(bag.tree2, newdata = data_final[-train, ])
#Confusion Matrix
table(yhat.bag, SATIS.test)
```

```
##         SATIS.test
## yhat.bag   0   1
##        0 355  67
##        1  24  85
```

```
# calculate misclassification rate
misclassification_rate <- mean(yhat.bag != SATIS.test)

# Adding to results table
df_results <- rbind(df_results, data.frame(
  Model = "Bagged Tree (50 trees)", Misclassification_rate = misclassification_rate))

cat("Misclassification Rate: ", misclassification_rate)
```

```
## Misclassification Rate:  0.1713748
```

Random Forest For random forests for classification, it is standard to use `mtry = root(p)` for the number of parameters (ISLR p358). So, since we have 57 predictors in total, we will use `mtry = root(57) = 8 (nearest integer)`.

```
rf.tree <- randomForest(SATIS ~ ., data = data_final,
                        subset = train, mtry = 8, importance = TRUE)
yhat.rf <- predict(rf.tree, newdata = data_final[-train, ])
#Confusion Matrix
table(yhat.rf, SATIS.test)
```

```
##        SATIS.test
## yhat.rf   0   1
##       0 365  70
##       1  14  82
```

```
# calculate misclassification rate
misclassification_rate <- mean(yhat.rf != SATIS.test)

# Adding to results table
df_results <- rbind(df_results, data.frame(
  Model = "Random Forest", Misclassification_rate = misclassification_rate))

# Print missclassification rate
cat("Misclassification Rate: ", misclassification_rate)
```

```
## Misclassification Rate:  0.1581921
```

```

# Get top 10 by MeanDecreaseGini
importance_table <- importance(rf.tree)
importance_table <- importance_table[order(importance_table[, "MeanDecreaseGini"],
                                           decreasing = TRUE), ]
importance_table <- head(importance_table, 10)

# Convert to data frame and include variable names as a column
importance_table <- as.data.frame(importance_table)
importance_table$Variable <- rownames(importance_table)
rownames(importance_table) <- NULL

# Keep only Variable and MeanDecreaseGini
importance_table <- importance_table[, c("Variable", "MeanDecreaseGini")]

# Number of columns
n_cols <- ncol(importance_table)

# Print table
kable(importance_table, format = "latex", booktabs = TRUE,
      caption = "Top 10 Variables by MeanDecreaseGini") %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(0, bold = TRUE, background = "#f9c5d1") %>%
  column_spec(1:n_cols, background = "#ffe6ec")
```

Interpretation Features that consistently reduce impurity a lot are likely to be more informative; they help the model make more accurate decisions. So, a higher MeanDecreaseGini implies that a feature is used often and makes meaningful splits, hence it is considered more important.

Table 1: Top 10 Variables by MeanDecreaseGini

Variable	MeanDecreaseGini
SDLIVING_CHILD	79.07843
COMMISSUE1_MOD_IMPUTED	49.64954
ORIGINBATTERY_a	47.75112
ST2_MOD	36.58494
INC_SDT1_MOD_RECODE	30.97490
ORIGINBATTERY_b	30.85538
ORIGINBATTERY_c	25.32390
BLK_Q30	24.83985
IMMVAL_MOD_a	24.39294
IMMVAL_MOD_b	22.24096

```
# Get top 10 by MeanDecreaseAccuracy
importance_table_acc <- importance(rf.tree)
importance_table_acc <- importance_table_acc[order(
  importance_table_acc[,"MeanDecreaseAccuracy"], decreasing = TRUE), ]
importance_table_acc <- head(importance_table_acc, 10)

# Convert to data frame and include variable names as a column
importance_table_acc <- as.data.frame(importance_table_acc)
importance_table_acc$Variable <- rownames(importance_table_acc)
rownames(importance_table_acc) <- NULL

# Keep only Variable and MeanDecreaseAccuracy
importance_table_acc <- importance_table_acc[, c("Variable", "MeanDecreaseAccuracy")]

# Number of columns
n_cols <- ncol(importance_table_acc)

# Print table
kable(importance_table_acc, format = "latex", booktabs = TRUE,
      caption = "Top 10 Variables by MeanDecreaseAccuracy") %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(0, bold = TRUE, background = "#f9c5d1") %>%
  column_spec(1:n_cols, background = "#ffe6ec")
```

This metric captures the actual contribution of a feature to the model's performance. It's often considered more reliable than MeanDecreaseGini because it's based on model prediction, not just internal splits.

Below are the features that score in the top 10 of importance for both metrics:

```
intersect(rownames(head(importance(rf.tree)[order(
  importance(rf.tree)[, "MeanDecreaseGini"], decreasing = TRUE), ], 10)),
  rownames(head(importance(rf.tree)[order(importance(
    rf.tree)[, "MeanDecreaseAccuracy"], decreasing = TRUE), ], 10)))
```

```
## [1] "SDLIVING_CHILD"         "COMMISSUE1_MOD_IMPUTED" "ORIGINBATTERY_a"
## [4] "ST2_MOD"                "INC_SDT1_MOD_RECODE"    "ORIGINBATTERY_b"
## [7] "ORIGINBATTERY_c"        "BLK_Q30"                "IMMVAL_MOD_a"
```

The random forest model indicates that the most informative predictors for our outcome variable are:

Table 2: Top 10 Variables by MeanDecreaseAccuracy

Variable	MeanDecreaseAccuracy
SDLIVING_CHILD	48.17362
ORIGINBATTERY_a	37.80426
COMMISSUE1_MOD_IMPUTED	32.59879
ST2_MOD	28.96059
ORIGINBATTERY_b	26.64630
IMMVAL_MOD_a	25.00877
ORIGINBATTERY_c	24.23721
INC_SDT1_MOD_RECODE	23.14688
BLK_Q30	23.08736
IMMVAL_MOD_e	22.39656

- How respondents feel their children's standard of living will compare to theirs when their children are the age the respondents are now.
- What respondents beleive is the most important issue facing the community they live in.
- Level of favourability of respondents' opinions of the U.S.
- The degree of change tp the U.S. immigration system that respondents think is necessary.
- Respondents' total family income from all sources, before taxes in the last year (2021).
- Level of favourability of respondents' opinions of China.
- How respondents think their standard of living compares to their parents when they were the same age.
- How important respondents feel increasing deportations of immigrants currently in the country illegally is as a goal for immigration policy in the U.S.
- How important respondents feel creating a way for most immigrants currently in the country illegally to stay here legally is as a goal for immigration policy in the U.S.

Gradient Descent

This model satisfies requirement number 2, using gradient descent. It also satisfies requirement number 5, sacrificing interpretability for the possibility of higher predictive accuracy.

Preproccessing and Standardisation

Preprocessing and standardisation were essential in implementing a stable and accurate gradient descent solution for logistic regression, and how these steps impacted model behavior and results.

We implemented and compared a logistic regression model using two approaches:

- **glm() with family = "binomial"** as a benchmark.
- **Gradient descent optimization** using numeric finite difference approximation of the gradient and the Barzilai–Borwein step size selection.

Without standardisation, both methods gave identical classification accuracy, yet their coefficient estimates were drastically different in scale, with gradient descent returning much larger magnitudes.The dataset was largely composed of categorical variables, which had, varying numbers of levels (e.g., some with 2 categories, others with 6 or more), correspondingly varied numbers of dummy variables and varying effective variances

depending on how evenly the categories were distributed. This led to some variables contributing much more to the gradient updates because, unlike glm(), which solves for coefficients analytically and is invariant to linear rescaling of features, gradient descent is sensitive to feature scale.

We standardised all features to ensure that each feature has mean zero and unit variance. all gradients are on comparable scales, no dummy variables dominate due to frequency and BB step size calculation becomes more stable and meaningful. After applying z-score standardization to the predictors the gradient descent coefficients closely matched those from glm()and the final model achieved the same predictive accuracy, but with coefficients on a comparable and meaningful scale.

```
# Convert predictors to numeric matrix (including dummy variables)
X_all <- data_final[ , setdiff(names(data_final), "SATIS")]
X_matrix <- data.matrix(X_all)

# Binary response as 0/1
Y_all <- as.numeric(as.character(data_final$SATIS))
Y_all <- ifelse(Y_all > 0.5, 1, 0)

# Train/test split using the same split as previous models
X_train_raw <- X_matrix[train, ]
X_test_raw  <- X_matrix[test, ]
Y_train <- Y_all[train]
Y_test  <- Y_all[test]

# Standardise based on training data
X_train_mean <- apply(X_train_raw, 2, mean)
X_train_sd   <- apply(X_train_raw, 2, sd)

X_train <- scale(X_train_raw, center = X_train_mean, scale = X_train_sd)
X_test  <- scale(X_test_raw,  center = X_train_mean, scale = X_train_sd)
```

Log-likelihood function and Numeric Differentiation

We used code from the week4gradientdescent.Rmd from the ST310 course materials to set up our gradient descent, making minor adjustments.

We added a step in the negative log-likelihood function to clamp `exp_ratio` to stay between a tiny positive value (eps) and 1 - eps, which prevents log(0) and keeps the loss function and gradients numerically stable and finite. Without clamping, our gradient descent was halting due to invalid values in the loss and gradients. By clamping the predicted probabilities within a small, safe range, we ensure that all logarithmic operations remain finite to enable smooth and reliable convergence during training.

```
neglogL <- function(X, Y, beta) {
  Xbeta <- X %*% beta
  expXbeta <- exp(-Xbeta)
  exp_ratio <- 1 / (1 + expXbeta)
  # clamping to avoid log(0) and log(1 - 1)
  eps <- 1e-8
  exp_ratio <- pmax(pmin(exp_ratio, 1 - eps), eps)
  -sum(Y * log(exp_ratio) + (1 - Y) * log(1 - exp_ratio))
}
```

We used the same numeric gradient function as in the course file, using the finite difference method to approximate the gradient as below. We found that no adjustments were needed.

$$\frac{f(x+h) - f(x-h)}{2h}$$

```
numeric_grad <- function(X, Y, beta, h = 1e-06) {
  # Approximate each coordinate of the gradient
  numerator <- sapply(1:length(beta), function(j) {
    H <- rep(0, length(beta))
    H[j] <- h # step in coordinate j
    neglogL(X, Y, beta + H) - neglogL(X, Y, beta - H)
  })
  numerator / (2*h)
}
```

As in the course materials, we will use the *Barzilai–Borwein method* for choosing a step size.

Initialise and take a first step:

```
max_steps <- 30
tol <- 1e-5 # (error) tolerance

# Training data using standardised data
X <- X_train
Y <- Y_train

# Initialise beta with correct length (number of predictors)
beta_prev2 <- rnorm(ncol(X))
grad_prev2 <- numeric_grad(X, Y, beta_prev2)

# First step (normalised gradient direction)
beta_prev1 <- beta_prev2 + 0.1 * grad_prev2 / sqrt(sum(grad_prev2^2))

# Compute the gradient
grad_prev1 <- numeric_grad(X, Y, beta_prev1)

# Initialise loss
previous_loss <- neglogL(X, Y, beta_prev2)
next_loss <- neglogL(X, Y, beta_prev1)
steps <- 1
```

Repeat until convergence:

```
while (abs(previous_loss - next_loss) > tol) {
  # Compute BB step size
  grad_diff <- grad_prev1 - grad_prev2
  step_BB <- sum((beta_prev1 - beta_prev2) * grad_diff) / sum(grad_diff^2)

  beta_prev2 <- beta_prev1

  # Update step
  beta_prev1 <- beta_prev1 - abs(step_BB) * grad_prev1

  # Shift previous steps
  grad_prev2 <- grad_prev1
  grad_prev1 <- numeric_grad(X, Y, beta_prev1)
  previous_loss <- next_loss
  next_loss <- neglogL(X, Y, beta_prev1)
```

```r
  # Print loss every 5 steps, track path, control loop
  if (round(steps / 5) == steps / 5) print(previous_loss)
  steps <- steps + 1
  beta_path[steps, 2] <- next_loss
  beta_path[steps, 3:ncol(beta_path)] <- beta_prev1
  if (steps > max_steps) break
}
```

```
## [1] 1969.368
## [1] 1273.498
## [1] 1233.399
## [1] 1224.297
## [1] 1223.176
## [1] 1222.912
```

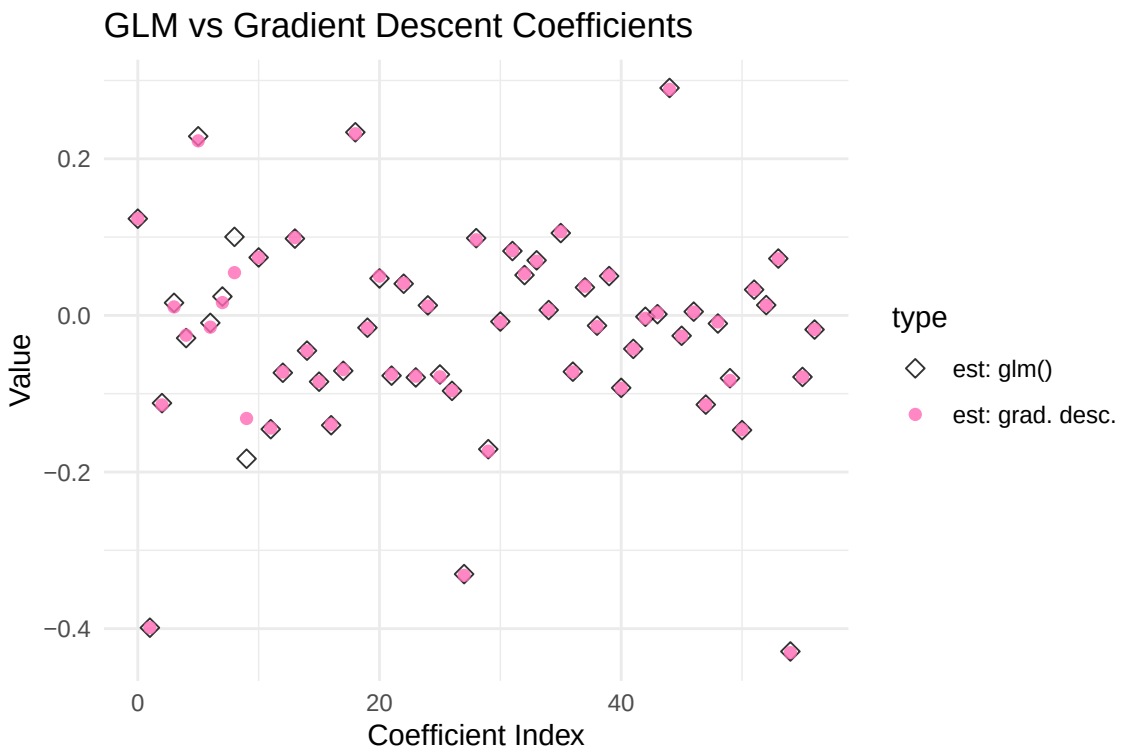
```r
cat("The number of steps taken is:", steps - 1, "\n")
```

```
## The number of steps taken is: 30
```

```r
# Fit the baseline logistic model using the standardised training data
train_df <- data.frame(SATIS = Y_train, X_train)
logistic <- glm(SATIS ~ . - 1, data = train_df, family = "binomial")
```

The plot below compares the estimate from our gradient descent implementation to the estimate from the `glm()` function for our baseline logistic regression model.

```r
p <- length(beta_prev1)
beta_plot_df <- data.frame(
  coordinate = 0:(p - 1),
  beta = c(coef(logistic), beta_prev1),
  type = c(rep("est: glm()", p), rep("est: grad. desc.", p))
)

ggplot(beta_plot_df, aes(x = coordinate, y = beta, color = type)) +
  geom_point(aes(shape = type), size = 2, alpha = 0.8) +
  scale_shape_manual(values = c(5, 16)) +
  scale_color_manual(values = c("est: glm()" = "black", "est: grad. desc." = "hotpink")) +
  theme_minimal() +
  labs(title = "GLM vs Gradient Descent Coefficients",
       x = "Coefficient Index", y = "Value")
```



The two sets of estimates align closely across all coefficient indices, indicating that our gradient descent successfully approximates the maximum likelihood estimates obtained from `glm()`. This suggests the gradient descent implementation is accurate and converged well.

Validating on the test set:

```
# Predict from glm()
pred_glm <- predict(logistic, newdata = as.data.frame(X_test), type = "response")

# Predict from gradient descent
pred_gd <- 1 / (1 + exp(-X_test %*% beta_prev1))

# Classify
pred_glm_class <- ifelse(pred_glm > 0.5, 1, 0)
pred_gd_class  <- ifelse(pred_gd  > 0.5, 1, 0)

# Accuracy
acc_glm <- mean(pred_glm_class == Y_test)
acc_gd  <- mean(pred_gd_class  == Y_test)

# Adding to results table
df_results <- rbind(df_results, data.frame(Model = "Gradient Descent",
                                           Misclassification_rate = 1- acc_gd))

cat("Accuracy (glm):", round(acc_glm, 4), "\n")
```

```
## Accuracy (glm): 0.6441
```

```
cat("Accuracy (grad. desc.):", round(acc_gd, 4), "\n")
```

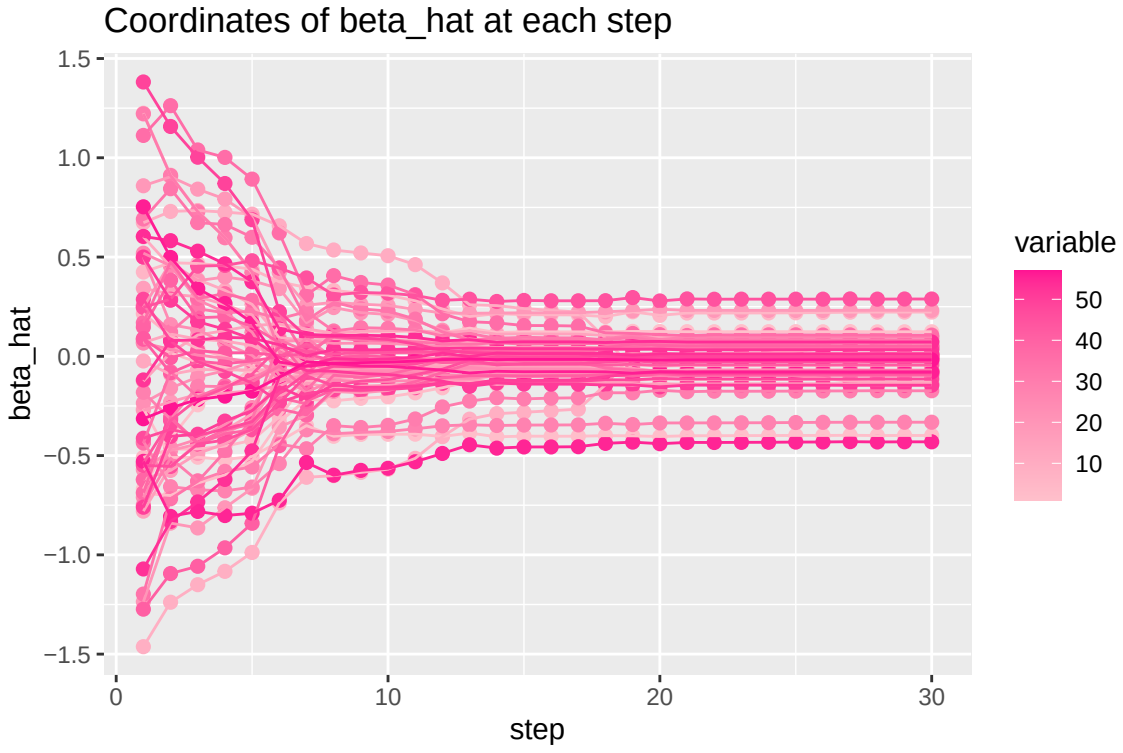
```
## Accuracy (grad. desc.): 0.6441
```

Our gradient descent implementation works correctly, converging to the same model fit as `logistic`. The matching classification accuracies prove predictive equivalence and the coefficient plot further confirms the equivalence of the two approaches.

We were also interested in our coefficient paths, function values and step distances. The plots below show our results.

Coefficient paths:

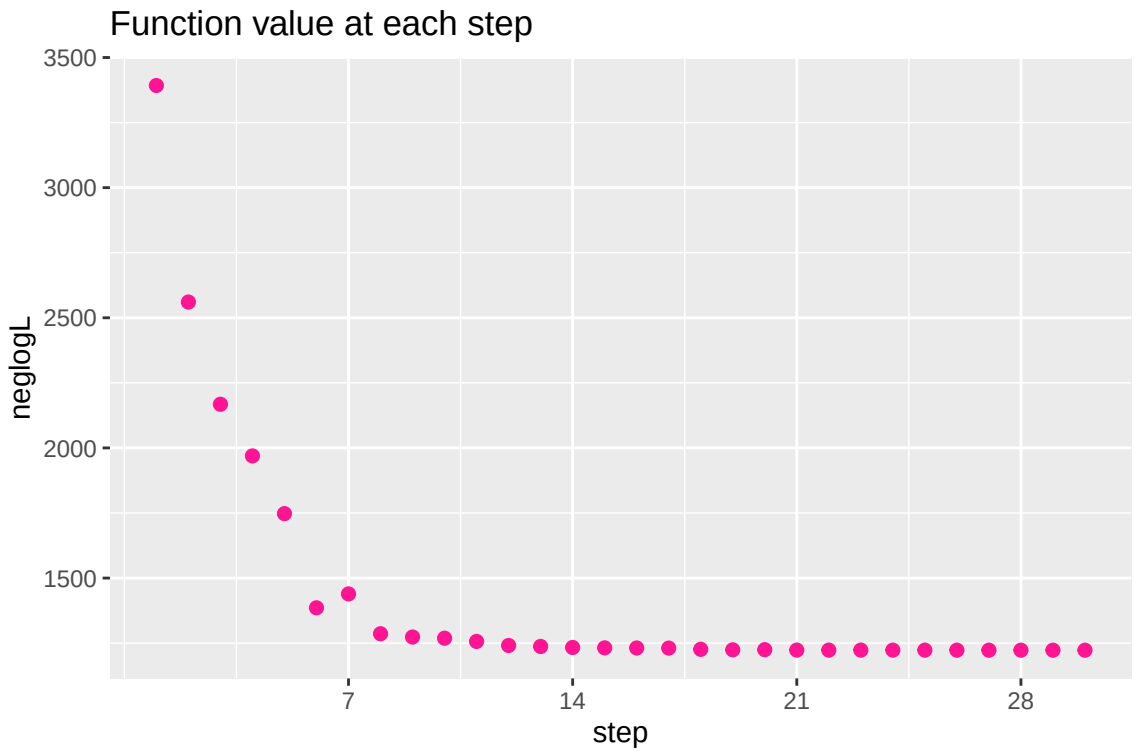
```
beta_long <- beta_path[-1, ] |>
  pivot_longer(cols = starts_with("X"), # Adapted to include only varriable columns
               names_to = "variable",
               values_to = "beta_hat") |>
  mutate(variable = as.numeric(gsub("X", "", variable)))

beta_long |>
  ggplot(aes(x = step,
             y = beta_hat,
             group = variable,
             color = variable)) +
  geom_point(size = 2) +
  geom_line() +
  scale_color_gradient(low = "pink", high = "deeppink") +
  ggtitle("Coordinates of beta_hat at each step")
```



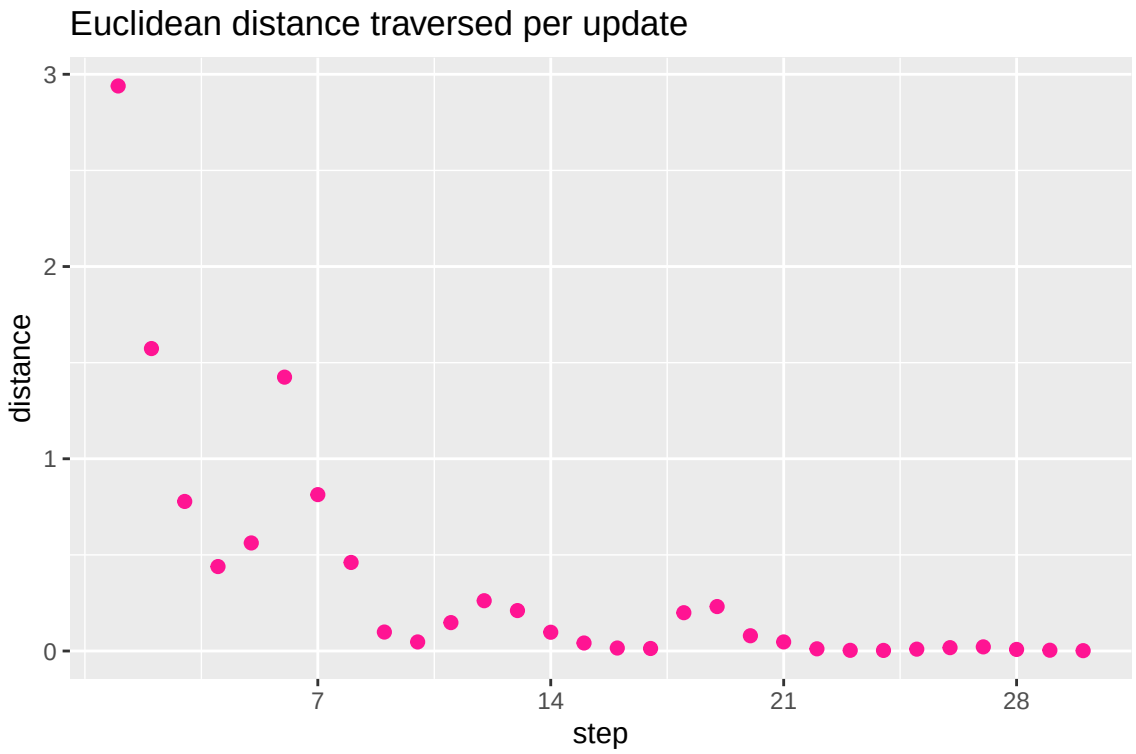
Values of the objective function:

```
beta_path[-1, ] |>
  ggplot(aes(step, neglogL)) +
  geom_point(size = 2, colour = "deeppink") +
  scale_x_continuous(breaks = (1:4) * floor(steps/4)) +
```

```
  ggtitle("Function value at each step")
```




```
distances <- sapply(2:steps, function(step) {
 sqrt(sum((beta_path[step, -c(1:2)] - beta_path[step-1, -c(1:2)])^2))
})
data.frame(step = 1:length(distances),
           distance = distances) |>
  ggplot(aes(step, distance)) +
  scale_x_continuous(breaks = (1:4) * floor(steps/4)) +
  geom_point(size = 2, colour = "deeppink") +
  ggtitle("Euclidean distance traversed per update")
```

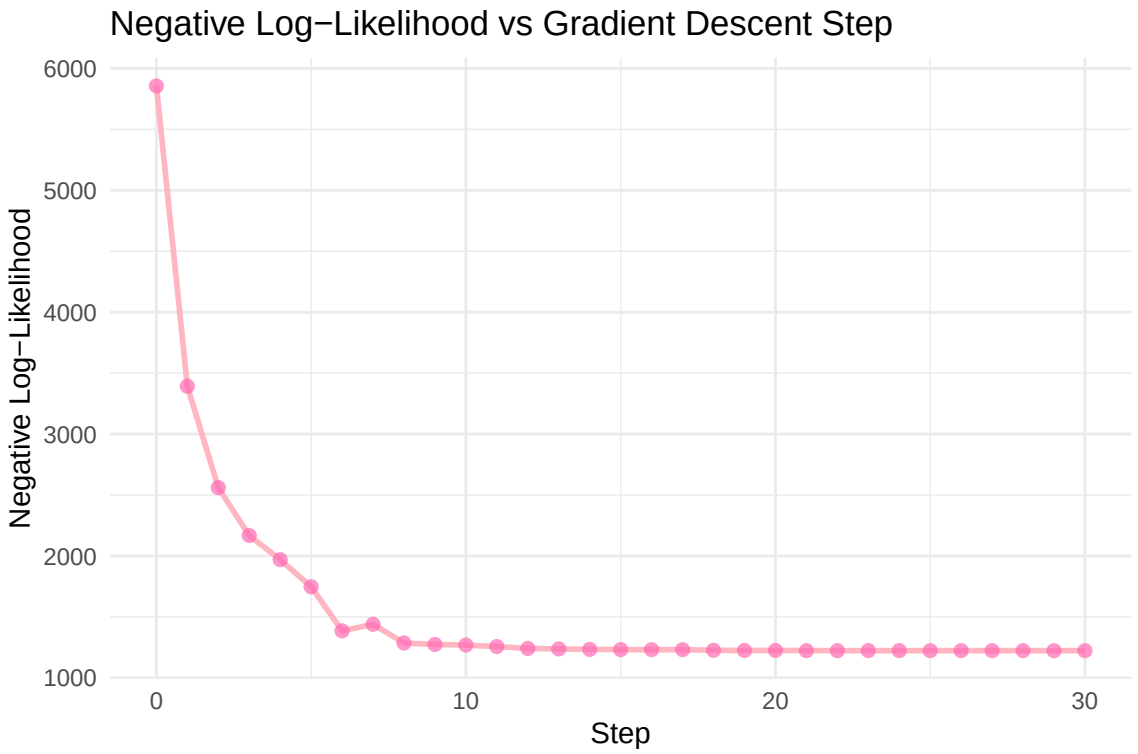


All plots show convergence at around 10 steps. Below, we futher explore the optimal number of steps to limit the model to.

Finding optimal maximum steps

To find the best values for `max_steps`, we explored the effect of different values on both the negative log-likelihood and coefficient change.

```
# Trim beta_path data to include only the number of steps used
beta_path_trimmed <- beta_path[1:steps, ]

# Plot negative log-likelihood vs steps of gradient descent
ggplot(beta_path_trimmed, aes(x = step, y = neglogL)) +
    geom_line(linewidth = 1, colour = "lightpink") +
    geom_point(size = 2, alpha = 0.7, colour = "hotpink") +
    theme_minimal() +
    labs(
        title = "Negative Log-Likelihood vs Gradient Descent Step",
        x = "Step",
        y = "Negative Log-Likelihood"
    )
```

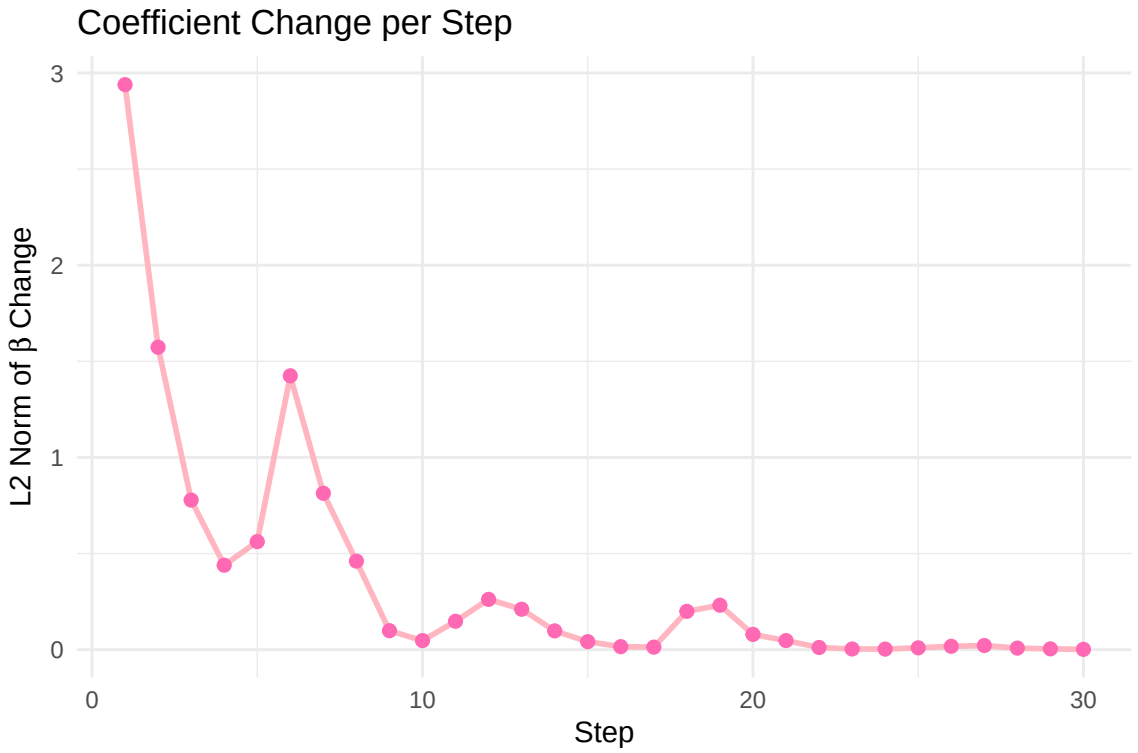


The above plot verifies convergence and shows that the curve flattens out at around step 10, showing convergence of the optimization. Beyond this point, additional steps don't significantly improve the model. The plot also verifies that the learning rate is appropriate and the model isn't underfitting or diverging. It does suggest however that training could potentially be stopped early (e.g., via early stopping around step 10). We tested a lower step size and found that the accuracies of `logistic` and gradient descent no longer matched, suggesting that coefficient convergence requires more steps.

The below plot shows the effect of step size on the change in the L2 norm of coefficients.

```r
# Calculate beta_diff
beta_diff <- apply(
  beta_path[2:steps, -(1:2)] - beta_path[1:(steps - 1), -(1:2)],
  1,
  function(row) sqrt(sum(row^2))
)

# Create a data frame
beta_diff_df <- data.frame(
  step = 1:(steps - 1),
  l2_norm = beta_diff
)

# Plot
ggplot(beta_diff_df, aes(x = step, y = l2_norm)) +
  geom_line(linewidth = 1, colour = "lightpink") +
  geom_point(size = 2, color = "hotpink") +
  theme_minimal() +
  labs(
    title = "Coefficient Change per Step",
    x = "Step",
```

```
    y = expression("L2 Norm of " * beta * " Change")
)
```



The plot shows that the coefficient change (L2 norm) drops sharply within the first 10 steps. This aligns with al our previous plots. From step 15 onward, the changes become very small, and after about step 20, they are likely due to numerical noise. Although the model converges numerically before step 30, we chose `max_steps = 30` to ensure both stable coefficients and stable predictions. Setting `max_steps` to 30 offers a good balance between computational efficiency and ensuring convergence. In our case, there is not much benefit to decreasing the maximum steps because the dataset is sufficiently small that computational efficiency is not a concern. Therefore, we decided it was best to keep the steps higher than potentially necessary to try and maximise predictive accuracy.

Penalised Logistic Regression (Lasso)

This model satisfies requirement number 4, being (relatively) high-dimensional.

Next we tested Lasso (or L2) penalised logistic regression (ISLR p274). This is ideal for high-dimensional settings because it simultaneously performs regularisation and feature selection, helping to build simpler, more robust, and more interpretable predictive models.

Dimensionality

Since our data is not very high-dimensional (only 57 predictor variables for 2653 observations), we increased the dimensionality of our data by adding pariwise interactions for every possible combination. This meant that, in addition to our 57 original predictors, we created 1,596 two-way interactions (57 choose 2). Due to our variables being categorical (with multiple levels), we ended up with a total of 8120 predictors. This satisfies the high-dimensional requirement since now $p > n$ ($8120 > 2653$). Due to the extremely high number of predictors now in the data, we use Lasso penalised logistic regression to perform automatic feature selection by shrinking the coefficients of irrelevant or redundant variables to exactly zero, excluding them from the

model.

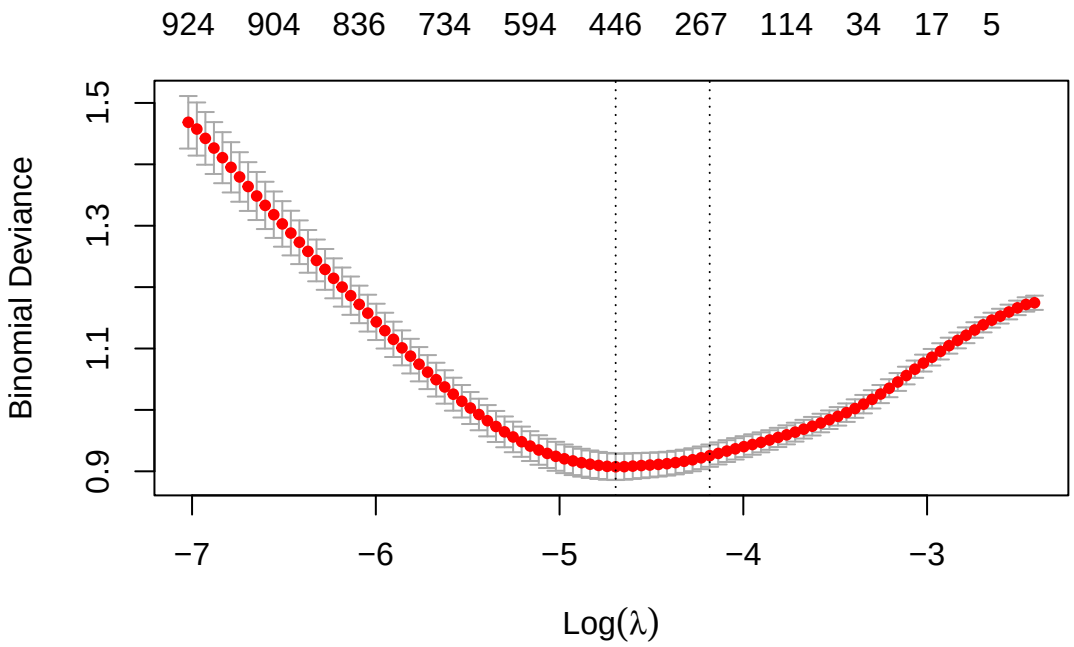
```
# Prepare data
y <- data_final$SATIS
X_data <- data_final %>% select(-SATIS) # Remove target from features

# Matrix of pairwise interactions
X_full <- model.matrix(~ (.)^2, data = X_data)[, -1]

# Check added dimensionality
cat("Original predictors:", ncol(X_data), "\n")
```

```
## Original predictors: 57
```

```
cat("Expanded feature space:", ncol(X_full), "\n")
```

```
## Expanded feature space: 8120
```

```
# Train and test split
X_train <- X_full[train, ]
X_test  <- X_full[test, ]
y_train <- y[train]
y_test  <- y[test]

# Fit Lasso logistic regression
cv_fit <- cv.glmnet(X_train, y_train, alpha = 1, family = "binomial")

# Plot cross-validation results to find best lamda
plot(cv_fit)
```




```
# Train final model using best lambda
best_lambda <- cv_fit$lambda.min
```

```r
model <- glmnet(X_train, y_train, alpha = 1, lambda = best_lambda, family = "binomial")

# Predict on test set
pred_prob <- predict(model, newx = X_test, type = "response")
pred_class <- ifelse(pred_prob > 0.5, 1, 0)

# Compute accuracy
accuracy <- mean(pred_class == y_test)
cat("Test Accuracy:", round(accuracy * 100, 2), "%\n")
```

```
## Test Accuracy: 80.6 %
```

```r
# Add to results table
df_results <- rbind(df_results, data.frame(Model = "Lasso",
                                           Misclassification_rate = 1- accuracy))

# Confusion matrix
cat("Confusion Matrix:\n")
```

```
## Confusion Matrix:
```

```r
print(table(Predicted = pred_class, Actual = y_test))
```

```
##           Actual
## Predicted   0   1
##         0 350  74
##         1  29  78
```

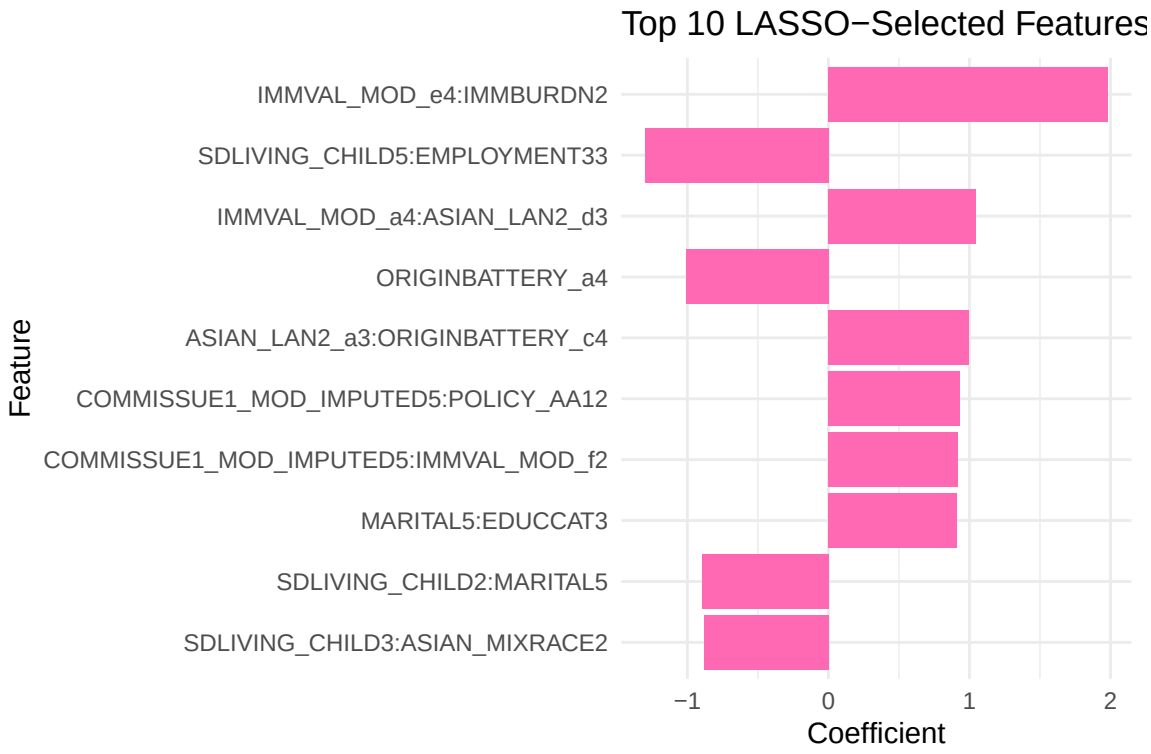
```r
# Number of selected (non-zero) coefficients
nonzero_coef <- coef(model)
cat("Number of selected features:", sum(nonzero_coef != 0), "\n")
```

```
## Number of selected features: 448
```

The resulting high-dimensional model has 448 predictors; reducing the number of features by roughly 94%. It performs well compared to the baseline logistic regression, the single tree models and gradient descent.

To see which features contribute most to predictions, we plotted the top ten features from the model with non-zero coefficients.

```r
# Convert to a tidy data frame and remove intercept
nonzero_df <- as.data.frame(as.matrix(nonzero_coef))
nonzero_df$feature <- rownames(nonzero_df)
colnames(nonzero_df)[1] <- "coefficient"
nonzero_df <- nonzero_df %>%
  filter(coefficient != 0 & feature != "(Intercept)") %>%
  arrange(desc(abs(coefficient)))

# Plot
ggplot(head(nonzero_df, 10), aes(x = reorder(feature, abs(coefficient)),
                                 y = coefficient)) + geom_col(fill = "hotpink") +
  coord_flip() +
  labs(title = "Top 10 LASSO-Selected Features",
       x = "Feature",
       y = "Coefficient") +
  theme_minimal()
```



The predictors with positive coefficients indicate that large values of these variables will push the outcome variable towards 1, or "Satisfied". The predictors with negative coefficients indicate that large values of these variables will push the outcome variable towards 0, or "Dissatisfied".

Some of these features seem totally counter-intuitive, for example, the top positive coefficient feature suggests that placing no importance on stricter immigration policies but believing that immigrants are a burden is a key indicator of feeling satisfied with the way things are going. This seems like an unlikely interaction of our original variables, so we explored further.

```
# Check frequency of suspicious interaction
table(
  data_final$IMMVAL_MOD_e == 4,
  data_final$IMMBURDN == 2
)
```

```
##
##         FALSE TRUE
##   FALSE  2149  338
##   TRUE    161    5
```

The table shows that only 5 respondents share this combination of attributes. Based on the high importance placed on the feature, we suspected that all 5 of these respondents had an outcome of `SATIS`, which we find below. This suggests the interaction may reflect overfitting to a small, consistent subgroup, rather than a generalisable pattern, highlighting the key issue of testing all interaction terms and using Lasso regression.

```
# Filter for those 5 people
subset_5 <- data_final$IMMVAL_MOD_e == 4 & data_final$IMMBURDN == 2

# View their SATIS values
table(data_final$SATIS[subset_5])
```

```
##
```

```
## 0 1
## 0 5
```

To summarise, while Lasso selected a rare interaction between contradictory views on immigration policy and immigrant burden, further inspection revealed that only 5 respondents held this combination, suggesting that this may be an overfit artifact rather than a robust signal.

The two methods we use to combat this are using `lambda.1se` to to automatically shrink weak or unstable signals and pre-filtering out rare interactions.

Feature Reduction

Lambda.1se

We decided to use largest value of `lambda` such that the cross-validated error is within 1 standard error of the minimum to automatically drop weak features, including rare interactions. This encourages stronger regularisation and fewer predictors

```
# Fit Lasso using lambda.1se
model_1se <- glmnet(X_train, y_train, alpha = 1, lambda = cv_fit$lambda.1se,
                    family = "binomial")

# See how many features are left
sum(coef(model_1se) != 0) - 1
```

```
## [1] 256
```

This shrunk the number of features to 285 (a 42% decrease).

```
# Predict on test set
pred_prob <- predict(model_1se, newx = X_test, type = "response")
pred_class <- ifelse(pred_prob > 0.5, 1, 0)

# Compute accuracy
accuracy <- mean(pred_class == y_test)
cat("Test Accuracy:", round(accuracy * 100, 2), "%\n")
```

```
## Test Accuracy: 77.02 %
```

```
# Add to results table
df_results <- rbind(df_results, data.frame(Model = "Lasso 1se",
                                           Misclassification_rate = 1- accuracy))

# Confusion matrix
cat("Confusion Matrix:\n")
```

```
## Confusion Matrix:
```

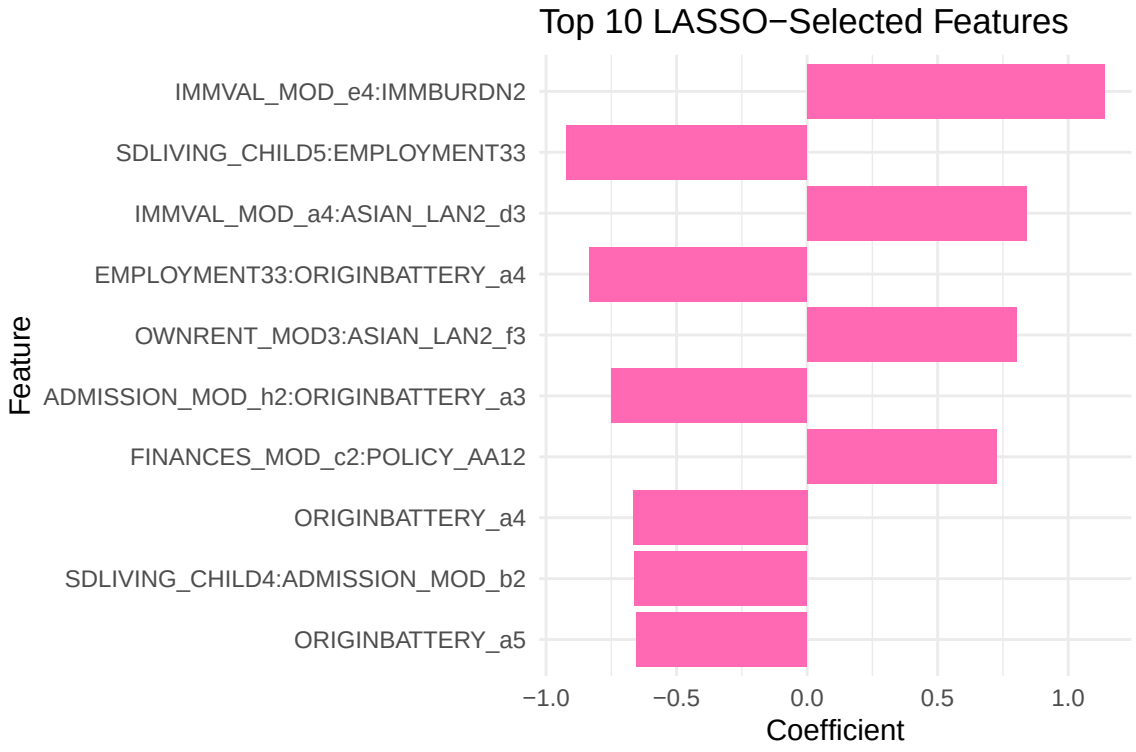
```
print(table(Predicted = pred_class, Actual = y_test))
```

```
##          Actual
## Predicted   0   1
##         0 352  95
##         1  27  57
```

```
# Number of selected (non-zero) coefficients
nonzero_coef <- coef(model_1se)
cat("Number of selected features:", sum(nonzero_coef != 0), "\n")
```

```
## Number of selected features: 257
```

```r
# Convert to a tidy data frame and remove intercept
nonzero_df <- as.data.frame(as.matrix(nonzero_coef))
nonzero_df$feature <- rownames(nonzero_df)
colnames(nonzero_df)[1] <- "coefficient"
nonzero_df <- nonzero_df %>%
  filter(coefficient != 0 & feature != "(Intercept)") %>%
  arrange(desc(abs(coefficient)))

# Plot
ggplot(head(nonzero_df, 10), aes(x = reorder(feature, abs(coefficient)), y = coefficient)) +
  geom_col(fill = "hotpink") +
  coord_flip() +
  labs(title = "Top 10 LASSO-Selected Features",
       x = "Feature",
       y = "Coefficient") +
  theme_minimal()
```



Pre-filtering

Now we try pre-filtering the data before fitting the model, to remove low frequency interactions that lead to overfitting when highly correlated to the outcome. We tried a few different thresholds, and settled on keeping only features with >= 100 non-zero entries, since this still retained 2839 features so still remains high-dimensional (p > n = 2653).

```r
# Count how many non-zero values each feature has
feature_freq <- colSums(X_full != 0)

# Set a threshold (we keep only features with >= 100 non-zero entries)
min_support <- 100
```

```r
X_filtered <- X_full[, feature_freq >= min_support]

# Compare dimensionality
cat("Original features:", ncol(X_full), "\n")
```

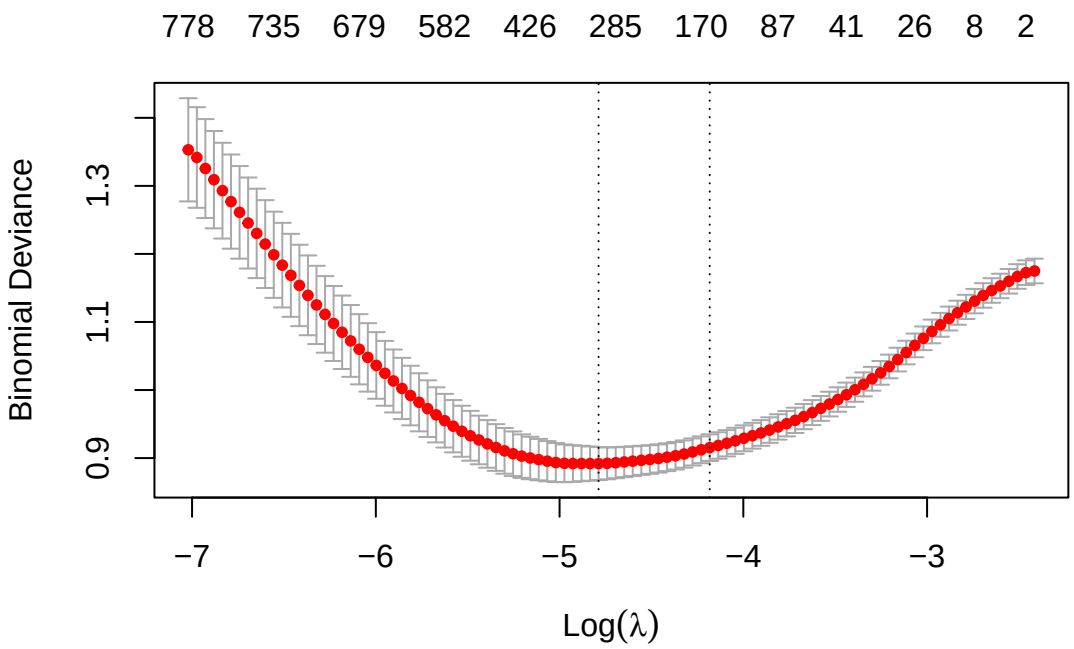
```
## Original features: 8120
```

```r
cat("After filtering low-frequency features:", ncol(X_filtered), "\n")
```

```
## After filtering low-frequency features: 3303
```

```r
# Update test/train split
X_train <- X_filtered[train, ]
X_test  <- X_filtered[test, ]
```

```r
# Fit Lasso logistic regression using pre-filtered data
cv_fit <- cv.glmnet(X_train, y_train, alpha = 1, family = "binomial")

# Plot cross-validation results to find best lamda
plot(cv_fit)
```




```r
# Train final model using best lambda
best_lambda <- cv_fit$lambda.min
model <- glmnet(X_train, y_train, alpha = 1, lambda = best_lambda,
                family = "binomial")

# Predict on test set
pred_prob <- predict(model, newx = X_test, type = "response")
pred_class <- ifelse(pred_prob > 0.5, 1, 0)

# Compute accuracy
```

```r
accuracy <- mean(pred_class == y_test)
cat("Test Accuracy:", round(accuracy * 100, 2), "%\n")
```

```
## Test Accuracy: 78.15 %
```

```r
# Add to results table
df_results <- rbind(df_results, data.frame(Model = "Lasso (filtered)",
                                           Misclassification_rate = 1- accuracy))

# Confusion matrix
cat("Confusion Matrix:\n")
```

```
## Confusion Matrix:
```

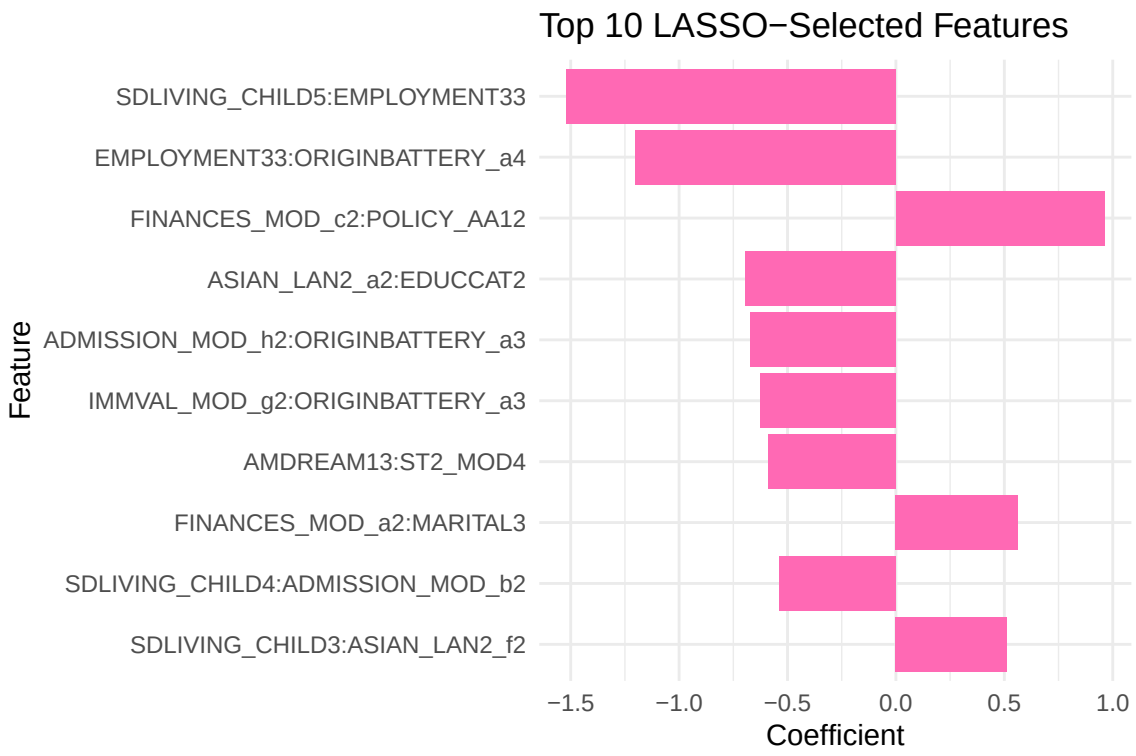
```r
print(table(Predicted = pred_class, Actual = y_test))
```

```
##           Actual
## Predicted   0   1
##         0 341  78
##         1  38  74
```

```r
# Number of selected (non-zero) coefficients
nonzero_coef <- coef(model)
cat("Number of selected features:", sum(nonzero_coef != 0), "\n")
```

```
## Number of selected features: 316
```

```r
# Convert to a tidy data frame and remove intercept
nonzero_df <- as.data.frame(as.matrix(nonzero_coef))
nonzero_df$feature <- rownames(nonzero_df)
colnames(nonzero_df)[1] <- "coefficient"
nonzero_df <- nonzero_df %>%
  filter(coefficient != 0 & feature != "(Intercept)") %>%
  arrange(desc(abs(coefficient)))

# Plot features
ggplot(head(nonzero_df, 10), aes(x = reorder(feature, abs(coefficient)), y = coefficient)) +
  geom_col(fill = "hotpink") +
  coord_flip() +
  labs(title = "Top 10 LASSO-Selected Features",
       x = "Feature",
       y = "Coefficient") +
  theme_minimal()
```



The top three features now are:

Top features for outcome "Unsatisfied":

1. When your children are at the age you are now, do you think their standard of living will be much worse than yours AND not enrolled in school.
2. Not enrolled in school and somewhat unfavourable opinion of the U.S.

Top for outcome "Satisfied":

3. No trouble paying bills in the past 12 months AND have not heard the phrase "affirmative action".

These features seem much more appropriate for the outcome. A small degree of predictive accuracy is sacrificed, but the higher accuracy of the original model was highly likely due to overfitting, as shown by the high importance of the extremely rare interaction.

Conclusion

The table below shows a summary of the accuracy of all models we tested.

```
kable(df_results, format = "latex", booktabs = TRUE,
      caption = "Results Table") %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(0, bold = TRUE, background = "#f9c5d1") %>%
  column_spec(1:ncol(df_results), background = "#ffe6ec")
```

The best model in terms of predictive accuracy is the random forest model. Our gradient descent performed particularly poorly, followed by our single tree models (both pruned and unpruned), then our penalised logistic regression. Random forest bagging methods performed well, but random forest using reduced predictors was the most accurate. This aligns with the findings from Oparina, E., Kaiser, C., Gentile, N. et al

Table 3: Results Table

Model	Misclassification_rate
Logistic Regression	0.2391714
Single Tree	0.2711864
Pruned Tree	0.2711864
Bagged Tree	0.1732580
Bagged Tree (50 trees)	0.1713748
Random Forest	0.1581921
Gradient Descent	0.3559322
Lasso	0.1939736
Lasso 1se	0.2297552
Lasso (filtered)	0.2184557

(2025) that tree-based models are better suited for predicting than linear models such as our baseline logistic regression.

Bibliography

AAPI Data (2024) Who are the AANHPI Voters in Swing States? https://aapidata.com/blog/who-are-the-aanhpi-voters-in-swing-states/

Brown and Sanders (2023) US Asians and Pacific Islanders view democracy with concern, AP-NORC/AAPI Data poll shows https://apnews.com/article/asian-americans-poll-democracy-biden-a7ca93b210c27ab237a3ca1dcf4042f3

Gallup (2025) Satisfaction With the United States https://news.gallup.com/poll/1669/general-mood-country.aspx

James, Witten, Hastie and Tibshirani (2023) An Introduction to Statistical Learning with Applications in R, Second Edition

Kennedy and Ruiz (2020) Polling methods are changing, but reporting the views of Asian Americans remains a challenge https://www.pewresearch.org/short-reads/2020/07/01/polling-methods-are-changing-but-reporting-the-views-of-asian-americans-remains-a-challenge/

Khan AE, Hasan MJ, Anjum H, Mohammed N, Momen S. (2024) Predicting life satisfaction using machine learning and explainable AI. Heliyon. 2024 May 20;10(10):e31158. https://pmc.ncbi.nlm.nih.gov/articles/PMC11137391/

MyAsianVoice (2024) The Asian American vote in the 2024 presidential election https://www.myasianvoice.com/the-asian-american-vote-in-the-2024-presidential-election

Oparina, E., Kaiser, C., Gentile, N. et al. Machine learning in the prediction of human wellbeing. Sci Rep 15, 1632 (2025). https://doi.org/10.1038/s41598-024-84137-1

Ramakrishnan and Fufunan (2024) Critical Battlegrounds for AANHPI Voter Impact in 2024 https://aapidata.com/featured/battlegrounds-aanhpi-2024/

Ruiz, Im and Tian (2023) Methodology: 2022-23 survey of Asian Americans https://www.pewresearch.org/race-and-ethnicity/2023/11/30/methodology-2022-23-survey-of-asian-americans/

Resources

'tree' package documentation

‘randomForest’ package documentation

ISLR2