An Ensemble Model to Classify Voter Propensity from Census Demographic Data

DA5030

Mark Apinis

Spring 2024

Contents

Introduction

In any political campaign, it is crucial to direct resources towards convincing the citizens most likely to vote. By creating a model that takes in some basic demographics and location data to predict if a voter will vote, everything from door knocking to get-out-the-vote texts can be optimized to target the people that are not likely voters.

In this project, we build three models to approach this problem: naive Bayes, logistic regression, and artificial neural networks. We then build an ensemble model to perform final voting classification through majority vote.

The data are sourced the Census's November 2022 Current Population Survey Voting and Registration Supplement, providing demographics and data for voters in the 2022 midterm elections.

Data Collection

Every even year, the Census Bureau adds a set of questions to its standard Current Population Survey (CPS). These questions relate to voting, covering topics such as if respondent voted in the most recent November elections, and if not, the reason why. Of course, not everyone is asked these questions, only those who were voting-age citizens at the time of the election are. Participation for each question, and the survey as a whole, was voluntary.

The source for this study is the Voting Supplement for the November 2022 elections. It has 127,097 observations of hundreds of features and is available for download in a variety of formats through the Census' website.

Data Translation

The native format for this data are Census microdata, a format that reduces file size by assigning specific character columns to each feature. The values for each feature are then stored as an integer within these characters. A vast majority of these features are categorical. The file does not contain a header line, instead feature positions and value translations are provided in a supplementary document.

The Census provides some tools to better handle this data. It is available for download as a CSV, but still uses the integers for feature values, not translated strings. This file is also much larger than the one needed for this work, as it includes many hundreds of features that are not being used in this analysis. The Census also has an online data compiler and table builder, but it supports too few features for this analysis.

With no good options, I chose to build my own tool to translate this microdata to CSV. It required building a manual dictionary for each feature's location and values, and so only some of the features are included. These features are the ones I chose to focus on for this analysis, such as basic demographics and location, so it is possible that an incredible predictor was missed. However, most of the features in this dataset are much more strongly related to jobs and economic strength, and would not be easy to obtain in a real-world use case of these models.

The features I chose to translate are:

- Age
- Sex
- Race (including mixed races)
- Marital status
- Family income
- State
- Metropolitan status (urban vs rural)
- Highest Education Achieved
- Citizenship (naturalized vs native-born)
- Student status

The code, dictionary, and November 2022 survey output are open source on GitHub.

Importing and Exploring Data

The CSV used in this analysis is hosted on the translation script's GitHub.

```r
csv <- read.csv(
  "https://raw.githubusercontent.com/mapinis/CPS-to-CSV/main/nov22pub.csv",
  stringsAsFactors = TRUE
)
```

From here, the data can start to be explored:

```
## 'data.frame':    126097 obs. of  11 variables:
##  $ family_income    : int  NA NA 50000 NA 50000 NA 35000 35000 60000 60000 ...
##  $ state            : Factor w/ 51 levels "AK","AL","AR",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ met_status       : Factor w/ 2 levels "metropolitan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ age              : int  -1 -1 45 -1 75 -1 77 85 65 72 ...
##  $ marital_status   : Factor w/ 6 levels "divorced","married_spouse_absent",..: NA NA 1 NA 6 NA 3 3
##  $ sex              : Factor w/ 2 levels "female","male": NA NA 2 NA 1 NA 1 2 2 1 ...
##  $ highest_education: Factor w/ 16 levels "10th_grade","11th_grade",..: NA NA 13 NA 11 NA 13 3 11 16
##  $ race             : Factor w/ 26 levels "AI-asian","AI-HP",..: NA NA 22 NA 22 NA 22 22 7 22 ...
##  $ citizenship      : Factor w/ 5 levels "native_born_abroad",..: NA NA 3 NA 3 NA 3 3 3 3 ...
##  $ student          : Factor w/ 2 levels "no","yes": NA NA 1 NA NA NA NA NA NA NA ...
##  $ voted            : Factor w/ 4 levels "dont_know","no",..: NA NA 2 NA 4 NA 4 4 4 4 ...
```

```
##  family_income        state                 met_status           age
##  Min.   :     0   CA     : 9901   metropolitan   :100500   Min.   :-1.00
##  1st Qu.: 35000   TX     : 7473   nonmetropolitan: 24274   1st Qu.: 5.00
##  Median : 60000   FL     : 6016   NA's           :  1323   Median :31.00
##  Mean   : 73425   NY     : 4860                            Mean   :32.67
##  3rd Qu.:100000   PA      : 3423                            3rd Qu.:57.00
##  Max.   :150000   IL      : 3343                            Max.   :85.00
##  NA's   :26360    (Other):91081
##                  marital_status      sex                highest_education
##  divorced               : 8543   female:51099   high_school       :23288
##  married_spouse_absent  : 1128   male  :48638   bachelor          :17326
##  married_spouse_present:41502   NA's  :26360   some_college      :12998
##  never_married          :24689                  master            : 7806
##  seperated              : 1339                  associate_academic: 4631
##  widowed                : 5493                  (Other)           :16645
##  NA's                   :43403                  NA's              :43403
##                 race                         citizenship    student
##  white          :79122   native_born_abroad  :  795   no  :39646
##  black          :10448   native_born_colonies:  432   yes : 7205
##  asian          : 5767   native_born_US      :86128   NA's:79246
##  american_indian: 1281   naturalized         : 6306
##  white-black    :  811   noncitizen          : 6076
##  (Other)        : 2308   NA's                :26360
##  NA's           :26360
##        voted
##  dont_know: 1527
##  no       :22393
##  refused  : 1286
##  yes      :39280
##  NA's     :61611
##
##
```

There are a total of 11 columns, one of which the target variable `voted`.

Data Preparation

The first step to analyzing this data and building models is preparation. This dataset is not built with this purpose in mind, so some modifications need to be done.

Filtering Data to Those Who Answered

The goal of the final ensemble model is binary classification: predicting whether someone will or won't vote. As a result, the dataset must only include cases where the respondent answered yes or no to the voting question.

```
voting_df <- csv[(csv$voted == "no" | csv$voted == "yes") & !is.na(csv$voted), ]
nrow(voting_df)
```

```
## [1] 61673
```

This removes about half of the rows.

Removing Incomplete Observations

The data now looks like this:

```
##  family_income        state                 met_status          age
##  Min.   :     0   CA      : 4711   metropolitan   :48706   Min.   :18.00
##  1st Qu.: 35000   TX      : 3177   nonmetropolitan:12313   1st Qu.:35.00
##  Median : 60000   FL      : 2600   NA's           :  654   Median :52.00
##  Mean   : 73804   NY      : 2133                           Mean   :51.25
##  3rd Qu.:100000   PA      : 1890                            3rd Qu.:67.00
##  Max.   :150000   OH      : 1598                            Max.   :85.00
##                   (Other):45564
##                  marital_status      sex                  highest_education
##  divorced              : 7008   female:32308   high_school         :17696
##  married_spouse_absent :  731   male  :29365   bachelor            :14042
##  married_spouse_present:32964                  some_college        :10514
##  never_married         :15484                  master              : 6284
##  seperated             :  932                  associate_academic  : 3822
##  widowed               : 4554                  associate_vocational: 2969
##                                                (Other)             : 6346
##                 race                     citizenship    student
##  white           :50773   native_born_abroad  :   587   no  :29736
##  black           : 6043   native_born_colonies:   309   yes : 3472
##  asian           : 2793   native_born_US      :55704   NA's:28465
##  american_indian :  695   naturalized         : 5073
##  white-AI        :  344   noncitizen          :     0
##  pacific_islander:  257
##  (Other)         :  768
##       voted
##  dont_know:    0
##  no       :22393
##  refused  :    0
##  yes      :39280
##
##
##
```

And the number of NA values in each column is:

```r
sapply(voting_df, function(c) sum(is.na(c)))
```

```
##     family_income             state        met_status               age
##                 0                 0               654                 0
##    marital_status               sex highest_education              race
##                 0                 0                 0                 0
##       citizenship           student             voted
##                 0             28465                 0
```

The `student` column has a significant number of NA values. Although this feature was initially believed to be useful, the high number of missing values meant this column should be dropped instead.

```r
voting_df$student <- NULL
```

The only other column with NA values is metropolitan status.

The proportions of voters to nonvoters in this NA sample is about equal to the overall proportions.

```r
prop.table(table(voting_df$voted))
```

```
##
## dont_know        no   refused       yes
## 0.0000000 0.3630924 0.0000000 0.6369076
```

```r
prop.table(table(voting_df$voted[is.na(voting_df$met_status)]))
```

```
##
## dont_know        no   refused       yes
## 0.0000000 0.4327217 0.0000000 0.5672783
```

So, to keep the dataset simple and NA-free, these rows are dropped.

```r
voting_df <- voting_df[!is.na(voting_df$met_status), ]
```

This leaves 61019 observations, with none more to be removed.

Special Feature Cases: Age and Citizenship

Although it may not seem it at first, the `age` feature is not purely continuous. According to the Census document, it is the exact age of the participant until 80, at which point 80-84 is recorded as 80, and 85+ as only 85. The reasoning behind this is not clear.

The feature will be treated as an integer scale, as the order of these categories has meanings. Because all of the dataset, including eventual test data, will be in the same format, no changes will be made here. However, it was important to note.

For citizenship, the data can be simplified into two categories, native born and naturalized. Citizenship origin is important, but with only 896 native-born respondents not born in the States, these observations would be better rolled into a general "native born" category.

```r
voting_df$citizenship <- factor(
  ifelse(voting_df$citizenship == "naturalized", "naturalized", "native_born")
)
summary(voting_df$citizenship)
```

```
## native_born naturalized
##       55974        5045
```

Outlier Analysis

We next perform outlier analysis on the only relevant features: family income and age. Although these features are scales, they still have a clear numeric order, and therefore outliers can be present. The specifics with age are explained above, and for family income, the value recorded is the bottom of the bin's range.

To perform this analysis, we define a function to return a boolean vector which says if the value in that row is more than three standard deviations away from the mean.

```r
# returns a vector of TRUE/FALSE values for if that row has an outlier
get_outliers <- function(x) {
  m <- mean(x)
  s <- sd(x)
  return(
    abs(x - m) / s >= 3
  )
}

# get the counts for the two rows
sapply(voting_df[, c("family_income", "age")], function(x) sum(get_outliers(x)))
```

```
## family_income           age
##             0             0
```

As shown above, there are no outliers for either of the features, so we move on.

Distribution Analysis

Here, a distribution analysis would be performed on the numeric features, family income and age. It is expected that family income would be normally distributed in the sample. Age, meanwhile, should not be, as age in the United States is not normally distributed to begin with (see Census brief here).

These tests would have been performed the standard way, with the built in Shapiro-Wilk normality test function. But, the `shapiro.test` function has a maximum sample size of 5000. Because of this, these tests could not be performed, as the sample size is here is over 60,000.

Encoding Data

Now that the data is pruned, empty levels can be dropped:

```r
factors <- which(sapply(voting_df, is.factor))

voting_df[, factors] <- lapply(
  voting_df[, factors],
  droplevels
)
```

Although different packages may be able to automatically handle categorical inputs, all of the models perform better or even need numerical features. Many of the features in this dataset are categorical, so they need to be encoded for use in training the models, specifically the regression.

Dummy encoding, where categories of n levels are split into $n - 1$ binary features, is simple and works well with all of these models. Here, a custom function is used that automatically dummy encodes the categories.

```r
# dummy encodes the specified column in the df
# grabs the levels and adds (num of factors) - 1 columns with 1s or 0s
# then drops the original col_name
# Side effect: if factor is all one level, it is just dropped
# returns a modified df
dummy_encode <- function(df, col_name) {
  # get the levels
  ls <- levels(df[, col_name])
  # loop over the first n-1 of them
  for (lev in ls[1:length(ls) - 1]) {
    # create column with appropriate values
    df[, paste(col_name, lev, sep = ".")] <- ifelse(df[, col_name] == lev, 1, 0)
  }
  # drop the column
  df[, col_name] <- NULL
  # return the modified df
  return(df)
}

# use reduce to dummy encode all categorical columns
voting_dummy <- Reduce(
  dummy_encode,
  setdiff(names(factors), c("voted")), # keep `voted` as-is
  voting_df
)
# as a last step, make col names syntactically valid
# some level names are not valid once they become column names
# `neuralnet` package complains otherwise
names(voting_dummy) <- make.names(names(voting_dummy), unique = TRUE)
```

A full `str` readout of columns in the new dataframe is available in the appendix.

Normalizing Data

As a final step for data preparation, the data are min-max scaled. This allows for better performance later on for the neural network.

```r
# use -3 for columns to exclude `voted`
voting_dummy[, -3] <- sapply(voting_dummy[, -3], function(x) {
  (x - min(x)) / (max(x) - min(x))
})
```

Splitting Data Into Training/Validation/Testing

The next step is to create the data splits that to use for model building and evaluation. A split of 64% training, 16% validation, and 20% testing was chosen based on applying a standard 80:20 ratio twice.

The ordering of the original data is not known, so it cannot be assumed to already be random. Thus, the splits are chosen by randomly sampling indices:

```r
train_idx <- sample(nrow(voting_dummy), 0.64 * nrow(voting_dummy))
valid_idx <- sample(
  setdiff(seq_len(nrow(voting_dummy)), train_idx), 0.16 * nrow(voting_dummy)
)
voting_train <- voting_dummy[train_idx, ]
voting_valid <- voting_dummy[valid_idx, ]
voting_test <- voting_dummy[c(-train_idx, -valid_idx), ]
```

Size of each group:

- Train: 39052
- Validate: 9763
- Test: 12204

Although this step is performed last, it does not violate any principles of test-group blindness, as all of the preparation steps occurred equally and irrespective of the `voted` value for an observation.

The Models

Three model types were chosen for individual analysis and final ensemble: naive Bayes, binomial logistic regression, and artificial neural networks. They were chosen because all of them perform well at binomial classification tasks and excel at modeling hidden relationships between features and the target.

They were also chosen because they represent different levels of algorithm complexity, from simple assumptions of independence and relative probabilities to black box methodology. The models will be trained and explored in increasing order of complexity to see how increasing complexity affects model performance.

Naive Bayes

The first model is naive Bayes. This model is simplest, it (naively) assumes independence between all of the features and compares relative probabilities.

The implementation used here is the `naive_bayes` function from the `naivebayes` package. Confusion matrices and performance metrics are from the `caret` package.

```r
m_nb <- naive_bayes(voted ~ ., data = voting_train, laplace = 0)
```

Building and Initial Evaluation We first build a model with no Laplace correction, despite it being unlikely that there is no accidental zero-ing out between these variables.

We evaluate this model using the validation dataset, using overall accuracy as a metric.

```r
pred <- predict(m_nb, newdata = voting_valid[, -3])
confusionMatrix(pred, voting_valid$voted, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1144  950
##        yes 2367 5302
##
##                Accuracy : 0.6602
##                  95% CI : (0.6508, 0.6696)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : 2.093e-05
##
##                   Kappa : 0.1908
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8480
##             Specificity : 0.3258
##          Pos Pred Value : 0.6914
##          Neg Pred Value : 0.5463
##              Prevalence : 0.6404
##          Detection Rate : 0.5431
##    Detection Prevalence : 0.7855
##       Balanced Accuracy : 0.5869
##
##        'Positive' Class : yes
##
```

With an overall accuracy of around 66%, this model is fair at most. But this accuracy figure hides the truly ugly specificity value: only 33%. When given the data for a respondent that did not vote, the model got it wrong more than half the time.

Improving This can hopefully be improved by adjusting the Laplace correction, removing the possibility of zeroing-out probabilities.

```r
m_nb2 <- naive_bayes(voted ~ ., data = voting_train, laplace = 1)
pred2 <- predict(m_nb2, newdata = voting_valid[, -3])
confusionMatrix(pred2, voting_valid$voted, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1144  950
##        yes 2367 5302
##
##                Accuracy : 0.6602
##                  95% CI : (0.6508, 0.6696)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : 2.093e-05
##
##                   Kappa : 0.1908
```

```
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8480
##             Specificity : 0.3258
##          Pos Pred Value : 0.6914
##          Neg Pred Value : 0.5463
##              Prevalence : 0.6404
##          Detection Rate : 0.5431
##    Detection Prevalence : 0.7855
##       Balanced Accuracy : 0.5869
##
##        'Positive' Class : yes
##
```

The change in Laplace correction led to no change in accuracy at all. In fact, every prediction made is the same:

```r
sum(pred != pred2)
```

```
## [1] 0
```

With no way to improve the model further, the second model is chosen to represent Naive Bayes, as the Laplace correction should in theory avoid a potential zero-out.

```r
m_nb <- m_nb2
```

Final Evaluation We now perform a final evaluation on this model using the test data. As metrics to compare the models across types, we use overall accuracy, specificity/sensitivity, kappa statistic, and Matthews correlation coefficient (MCC).

We now define a model evaluation function:

```r
# compares model predictions to voting_test and prints statistics
# newdata is an argument because naivebayes class
#   is particular about including target column
# pred_function is a custom function that returns the model's predicted values
evaluate_model <- function(
    pred_function,
    newdata = voting_test) {
  pred <- pred_function(newdata)
  cm <- confusionMatrix(pred, voting_test$voted, positive = "yes")
  mcc <- cor(pred == "yes", voting_test$voted == "yes")

  print(cm$table)
  cat("\n")
  cat("Accuracy:", cm$overall["Accuracy"], "\n")
  cat("\tSensitivity", cm$byClass["Sensitivity"], "\n")
  cat("\tSpecificity", cm$byClass["Specificity"], "\n")
  cat("Kappa:", cm$overall["Kappa"], "\n")
  cat("MCC:", mcc)
}
```

And use it to evaluate our naive Bayes model:

```r
nb_pred_function <- function(nd) predict(m_nb, newdata = nd)
evaluate_model(nb_pred_function, newdata = voting_test[, -3])
```

```
##           Reference
## Prediction   no  yes
##        no  1402 1209
##        yes 2985 6608
##
## Accuracy: 0.6563422
##  Sensitivity 0.8453371
##  Specificity 0.3195806
## Kappa: 0.180993
## MCC: 0.1929703
```

Not much different from the validation set. A barely passable accuracy of 65.6%, with okay sensitivity but horrible specificity. But, for the real-world use case, it may be better to have the model biased in this way. When targeting campaign outreach or advertisements, it is probably better to accidentally target non-voters instead of missing people who are likely to vote. However, the sensitivity of around 85% is not exactly great for this task either.

As expected, the kappa statistic and MCC reflect this, with the former calculating "poor agreement" between predicted and actual and the latter describing the model as "weakly correct", according to descriptions provided in Brett Lantz's *Machine Learning with R, Fourth Edition*, cited below.

Binomial Logistic Regression

The next model is binomial logistic regression. Unlike naive Bayes, it does not assume independence among features, making it more likely to capture overlapping interactions. However, it does assume that that the relationship between features and the log-odds of the target is linear.

For this model, we first build a model with all features, evaluate it, and then attempt to improve upon it through backward feature elimination.

Here, the built-in `glm` function is used.

```r
m_reg <- glm(voted ~ ., data = voting_train, family = binomial(link = "logit"))
pred_probs <- predict(m_reg, newdata = voting_valid, type = "response")
pred <- factor(ifelse(pred_probs > 0.5, "yes", "no"))
confusionMatrix(pred, voting_valid$voted, positive = "yes")
```

Building and Initial Evaluation

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1746  961
##        yes 1765 5291
##
##                Accuracy : 0.7208
##                  95% CI : (0.7118, 0.7297)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3617
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8463
##             Specificity : 0.4973
##          Pos Pred Value : 0.7499
##          Neg Pred Value : 0.6450
##              Prevalence : 0.6404
##          Detection Rate : 0.5419
##    Detection Prevalence : 0.7227
##       Balanced Accuracy : 0.6718
##
##        'Positive' Class : yes
##
```

As we can see, this model already performs much better than either of the naive Bayes models, with a large gain in specificity leading to a higher overall accuracy. Sensitivity remained similar to naive Bayes.

Improving Now, we aim to improve this model through backwards elimination of features that have a high p-value. These features do not effect to the target probabilities enough to reject the null hypothesis, which is that the feature is not relevant to prediction.

We perform this by using `summary` to find the feature p-values and then filtering to those with a p-value less than or equal to 0.05.

```r
significant <- Filter(function(p) p <= 0.05, summary(m_reg)$coefficients[, 4])
# summary$coefficients[, 4] is a list of p-values for each feature
# build a formula from the significant names, dropping "Intercept"
fmla <- as.formula(
  paste("voted ~ ", paste(names(significant[-1]), collapse = "+"))
)
fmla
```

```
## voted ~ family_income + age + state.AK + state.AZ + state.CA +
##     state.CO + state.DC + state.FL + state.GA + state.HI + state.IL +
##     state.KS + state.KY + state.LA + state.MA + state.MD + state.ME +
##     state.MI + state.MN + state.MO + state.MT + state.NC + state.NH +
##     state.NM + state.NV + state.NY + state.OR + state.PA + state.RI +
##     state.SD + state.UT + state.VA + state.VT + state.WA + state.WI +
##     marital_status.divorced + marital_status.married_spouse_present +
##     marital_status.never_married + sex.female + highest_education.10th_grade +
##     highest_education.11th_grade + highest_education.12th_grade +
##     highest_education.1st_grade + highest_education.4th_grade +
##     highest_education.6th_grade + highest_education.8th_grade +
##     highest_education.9th_grade + highest_education.bachelor +
##     highest_education.doctorate + highest_education.high_school +
##     highest_education.master + highest_education.professional +
##     race.american_indian + race.black + race.W.A.HP + race.white +
##     race.white.AI + race.white.asian + race.white.black + citizenship.native_born
```

With our new formula, we build and evaluate our next model:

```r
m_reg2 <- glm(fmla, data = voting_train, family = binomial(link = "logit"))
pred_probs <- predict(m_reg2, newdata = voting_valid, type = "response")
pred <- factor(ifelse(pred_probs > 0.5, "yes", "no"))
confusionMatrix(pred, voting_valid$voted, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1754  950
##        yes 1757 5302
##
##                Accuracy : 0.7227
##                  95% CI : (0.7137, 0.7316)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3661
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8480
##             Specificity : 0.4996
##          Pos Pred Value : 0.7511
##          Neg Pred Value : 0.6487
##              Prevalence : 0.6404
##          Detection Rate : 0.5431
##    Detection Prevalence : 0.7230
##       Balanced Accuracy : 0.6738
##
##        'Positive' Class : yes
##
```

A very small increase in accuracy, but still an increase. As p-values change with the number of features, we check if the new model has any features whose p-value increased and can be eliminated in the next round:

```r
sum(summary(m_reg2)$coefficients[, 4] > 0.05)
```

```
## [1] 0
```

And no it does not! All of the features in this second model are significant with an α of 0.05.

Therefore, we stop model iteration here, and assign our final regression model to the improved version.

```r
m_reg <- m_reg2
```

Final Evaluation Now, we evaluate the final binomial logistic regression model with the test data. To do this, we use the same evaluation function as was used for naive Bayes.

```r
reg_pred_function <- function(nd) {
  probs <- predict(m_reg, newdata = nd, type = "response")
  return(factor(ifelse(probs > 0.5, "yes", "no")))
}
evaluate_model(reg_pred_function)
```

```
##           Reference
## Prediction   no  yes
##        no  2170 1158
##        yes 2217 6659
##
## Accuracy: 0.7234513
##  Sensitivity 0.8518613
##  Specificity 0.4946433
## Kappa: 0.3658819
## MCC: 0.3733469
```

Similar to the smaller evaluations on the validation dataset, the final regression model has an accuracy of around 72%. Compared to naive bayes, the sensitivity is about the same, but specificity increases to nearly 50%, which although not great, drives the increase in overall accuracy.

The Kappa statistic and MCC increased in turn, upgrading to "fair agreement" and "moderately correct" according to the scales provided by Lantz.

The increases in all measures of model performance leads us to conclude that binomial logistic regression better predicts voter propensity than naive Bayes.

Artificial Neural Network

Lastly, we build an Artificial Neural Network (ANN). This is the most advanced model out of the three, free of the assumptions of feature independence or linear relationships. This allows the model to reflect very complex relationships between the features and target, recognizing and focusing on patterns that a human may not even notice.

However, it is a black-box model. While it would be possible to more deeply analyze the naive Bayes or logistic regression models by looking at probability tables or coefficients, ANNs only have the weights it uses to compare node values, and with any more than a few features and nodes, it is nearly impossible for a person to fully understand all of the interactions. And, through backpropagation of errors, these weights are decided by mathematical patterns, not natural ones.

In this analysis, we use the neural network implementation in the `neuralnet` package, with the default RPROP+ (resilient backpropagation) algorithm.

Building For the first model, we start with one hidden layer of 10 nodes.

As an activation function, we use the default logistic function. As it is a sigmoid function, it performs best for binary classification tasks, according to some basic research. Now, we train the model:

```r
m_ann <- neuralnet(
  voted == "yes" ~ ., # ensures one output node of probability
  data = voting_train,
  hidden = 10,
  stepmax = 1e6,
  linear.output = FALSE # neuralnet documentation for binary classification
)
```

With so many features, any plot of nodes is unreadable, but we can print the sum-of-squares error and number of steps taken to compute:

```
##     error     steps
##   3203.62 120212.00
```

And we can determine preliminary accuracy by building a prediction for the validation dataset:

```r
pred_probs <- predict(m_ann, newdata = voting_valid)
pred <- factor(ifelse(pred_probs > 0.5, "yes", "no"))
confusionMatrix(pred, voting_valid$voted, positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1759 1030
##        yes 1752 5222
##
##                Accuracy : 0.715
##                  95% CI : (0.706, 0.724)
##     No Information Rate : 0.6404
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3521
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8353
##             Specificity : 0.5010
##          Pos Pred Value : 0.7488
##          Neg Pred Value : 0.6307
##              Prevalence : 0.6404
##          Detection Rate : 0.5349
##    Detection Prevalence : 0.7143
##       Balanced Accuracy : 0.6681
##
##        'Positive' Class : yes
##
```

The confusion matrix above shows that this neural network performs about as well as the regression model, despite the exponential increase in computing time and setup complexity. An interesting pattern to note is that all of the models have a "yes" bias, incorrectly classifying at least half of the "no" respondents, leading to a low specificity. However, this is not counterbalanced by an incredibly high sensitivity.

Nevertheless, this is only the first iteration of the neural network.

Improving This model can potentially be improved in several ways. The first is by experimenting with the different activation functions, such as tanh, softplus, or other varieties of the sigmoid function. These changes affect how the model learns and finds the final output node values from the input features. A potentially more effective strategy, especially compared to the first iteration, is to instead increase the model's capacity to learn. Each node in the hidden layer, and the weights between it and the features, represents a pattern that the model training notices. By increasing the number of hidden nodes, more patterns can be learned.

This effect is essentially multiplied when the number of hidden layers is increased too, allowing the model to learn "patterns among the patterns." The first attempts to improve the model involved increasing the number of hidden layers, first with two layers of ten nodes each, then ten and five, and finally five and five. However, with the increase in learning comes a similarly multiplicative increase in training difficulty. For all three of these attempted models, model training was still ongoing after 15 hours on an M3 Pro, at which point it was aborted.

As a last attempt, a model with only one hidden layer but with 20 nodes was tried, but also stopped after 5 hours due to the project deadline. ANN training, at least in R, is not multithreaded, contributing to the processing difficulty. But the most important factor was likely the pure scale of the data: around 40,000 observations of nearly 100 features makes each training step (out of a million) incredibly difficult.

Final Evaluation And like the previous two models, we now perform a final evaluation on the ANN using the evaluation function.

```r
ann_pred_function <- function(nd) {
  probs <- predict(m_ann, newdata = nd)
  return(factor(ifelse(probs > 0.5, "yes", "no")))
}
evaluate_model(ann_pred_function)
```

```
##           Reference
## Prediction   no  yes
##        no  2067 1313
##        yes 2320 6504
##
## Accuracy: 0.7023107
##  Sensitivity 0.8320327
##  Specificity 0.4711648
## Kappa: 0.3192763
## MCC: 0.3251161
```

In the end, this model actually performs worse than the logistic regression, with only a 70.2% accuracy. And, just like naive Bayes and the regression, there is a "yes" bias, leading to a much higher sensitivity than specificity.

As expected, Kappa statistic and MCC are also lower, but still in the "fair agreement" and "moderately correct" Lantz categories.

Ensemble Model

With the model assemblies complete, we can now use the models and prediction functions to create an overall ensemble model. This model will predict the potential voter classification using all three models, and then return the result that the majority of the models agree on.

The "majority rules" ensemble model was chosen because no model is significantly enough better than another to warrant unequal weighting. At first, it was considered that if one model particularly better at predicting a certain outcome, such as if the ANN had really strong sensitivity, it would have a greater weight or even a veto during only those cases, but these results show very little difference between the models' performance.

Building

```r
ensemble_predict <- function(nd) {
  nb_pred <- nb_pred_function(nd[, -3])
  reg_pred <- reg_pred_function(nd)
  ann_pred <- ann_pred_function(nd)

  # decide from majority
  return(factor(ifelse(
    nb_pred == reg_pred,
    nb_pred,
    ann_pred
  ), labels = c("no", "yes")))
}
```

Final Evaluation

With the ensemble prediction function built, we can, for the final time, perform an evaluation with the test data:

```r
evaluate_model(ensemble_predict)
```

```
##           Reference
## Prediction   no  yes
##        no  1981 1095
##        yes 2406 6722
##
## Accuracy: 0.7131268
##  Sensitivity 0.8599207
##  Specificity 0.4515614
## Kappa: 0.3333367
## MCC: 0.3442358
```

In the end, this ensemble model is not impressive. It actually has a lower overall accuracy than the logistic regression model, and the common issue of false positives is retained. The other metrics are unchanged: the Kappa statistic implies a "fair agreement" between predictions and reality, and the MCC says the predictions are "moderately correct."

Conclusion

In the end, the model results are somewhat underwhelming. Although there was a meaningful increase in model accuracy from naive Bayes to binomial logistic regression, model improvements stopped there, ending with around 72% accuracy. Additionally, the problems with low specificity persisted, even through to the ensemble model. This relative lack of difference between the ensemble and individual models seems to imply that the models' failures occur for the same test cases. There may be some combination of feature values where these models consistently fail, and in those cases agree on the wrong prediction.

More work needs to be done in this field, but until then, this ensemble model may still be somewhat useful to make predictions for door-knocking.

References

[1] L. Blakeslee, Z. Caplan, J. A. Meyer, M. A. Rabe, and A. W. Roberts, "Age and Sex Composition: 2020," 2020 Census Briefs, May 2023, https://www2.census.gov/library/publications/decennial/2020/census-briefs/c2020br-06.pdf (accessed April 14, 2024).

[2] J. Brownlee. "How to Choose an Activation Function for Deep Learning," 2021, https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/ (accessed April 12, 2024).

[3] T. Challenor, "Predicting Votes from Census Data,", CS229, Stanford University, 2017, https://cs229.stanford.edu/proj2017/final-reports/5232542.pdf (accessed April 4, 2024).

[4] S. Fritsch, F. Guenther, M. N. Wright, M. Suling, and S. M. Mueller, "Package 'neuralnet'," CRAN, 2022, https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf (accessed April 13, 2024).

[5] B. Lantz, Machine Learning with R, 4 ed.: PACKT Publishing Limited, 2023.

[6] United States Census Bureau (2023). CURRENT POPULATION SURVEY, NOVEMBER 2022 VOTING SUPPLEMENT FILE. [Online] Available: https://www2.census.gov/programs-surveys/cps/techdocs/cpsnov22.pdf

[7] United States Census Bureau. "Voting and Registration," 2023, https://www.census.gov/topics/public-sector/voting.html (accessed April 2, 2024).

Appendix

Features in dummy-encoded voter data

```
## 'data.frame':    61019 obs. of  99 variables:
##  $ family_income                        : num  0.333 0.333 0.233 0.233 0.4 ...
##  $ age                                  : num  0.403 0.851 0.881 1 0.701 ...
##  $ voted                                : Factor w/ 2 levels "no","yes": 1 2 2 2 2 2 2 2 2 1 ...
##  $ state.AK                             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.AL                             : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ state.AR                             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.AZ                             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.CA                             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.CO                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.CT                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.DC                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.DE                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.FL                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.GA                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.HI                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.IA                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.ID                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.IL                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.IN                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.KS                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.KY                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.LA                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.MA                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.MD                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.ME                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.MI                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.MN                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.MO                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.MS                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.MT                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.NC                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.ND                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.NE                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.NH                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.NJ                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.NM                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.NV                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.NY                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.OH                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.OK                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.OR                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.PA                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.RI                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.SC                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.SD                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.TN                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.TX                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.UT                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.VA                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.VT                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.WA                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.WI                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ state.WV                              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ met_status.metropolitan               : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ marital_status.divorced               : num  1 0 0 0 1 0 0 0 0 0 ...
##  $ marital_status.married_spouse_absent  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ marital_status.married_spouse_present : num  0 0 1 1 0 1 1 1 0 0 ...
##  $ marital_status.never_married          : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ marital_status.seperated              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ sex.female                            : num  0 1 1 0 0 1 0 0 1 0 ...
##  $ highest_education.10th_grade          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ highest_education.11th_grade          : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.12th_grade          : num  0 0 0 1 0 0 0 0 0 0 ...
## $ highest_education.1st_grade           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.4th_grade           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.6th_grade           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.8th_grade           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.9th_grade           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.associate_academic  : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.associate_vocational: num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.bachelor            : num  0 1 0 0 1 0 1 1 0 0 ...
## $ highest_education.doctorate           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.high_school         : num  1 0 1 0 0 0 0 0 0 0 ...
## $ highest_education.master              : num  0 0 0 0 0 0 0 0 0 0 ...
## $ highest_education.professional        : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.AI.asian                         : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.AI.HP                            : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.american_indian                  : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.asian                            : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.asian.HP                         : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.B.AI.A                           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.black                            : num  0 0 0 0 1 0 0 0 0 0 ...
## $ race.black.AI                         : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.black.asian                      : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.black.HP                         : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.other_4_or_more                  : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.pacific_islander                 : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.W.A.HP                           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.W.AI.HP                          : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.W.AI.W                           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.W.B.A                            : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.W.B.AI                           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.W.B.AI.A                         : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.W.B.HP                           : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.white                            : num  1 1 1 1 0 1 1 1 1 1 ...
## $ race.white.AI                         : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.white.asian                      : num  0 0 0 0 0 0 0 0 0 0 ...
## $ race.white.black                      : num  0 0 0 0 0 0 0 0 0 0 ...
## $ citizenship.native_born               : num  1 1 1 1 1 1 1 1 1 1 ...
```