Predicting the Votes of the 1964 Civil Rights Act

ArchibaldDC

14/05/2020

Table of Contents

Introduction

One of the most defining legislations of the 1960s is the Civil Rights Act of 1964, aimed at outlawing discrimination based on race, color, religion, sex, or national origin. According to many historians in American history, it also marks a split in the history of the Democratic and Republican party. Simplified for the purpose of this project, the Civil Rights Act of 1964, is one of the root causes of the Republican Party's "Southern Strategy", which aimed to increase political support for the GOP among white Democratic voters who mainly opposed Northern Democrats and Republicans'views on civil rights. Passed under the Democratic administration of President Johnson, the Civil Rights act of 1964 pushed many African-Americans to vote Democrat in the upcoming elections, which directed the party towards a progressive direction and thus alienating its considerable conservative white voter base in the South.

Though history is more complex than that, the context is important to understand the goal of this project.

Goal of the Project

The bill was introduced in the House of Representatives in the summer of 1963 and signed into law by President Johnson a year later, all this was enacted during the 88th Congress.

Through machine learning, this project aims to predict the votes of the members from the 88th Congress' House of Representatives based on every house member's ideology estimation gathered on the Voteview website. Voteview permits users to check every congressional rol call vote on a economic-liberal vs. conservative ideological map.

Ideology is based on two parameters, Nominate and Nokken-Poole estimates, with two dimensions each. the variables are written as follow.

- nominate_dim1
- nominate_dim2
- nokken_poole_dim1
- nokken_poole_dim2

The Nominate estimate assumes that a Congressman has the same ideological position throughout his career. Nokken-Poole estimates are based on the assumption that each Congress is separate in terms of its members' poltical ideology.

The first dimension is the familiar economic-liberalism vs. conservative approach to politics, the second dimension, is based on the more critical, pertinant issues of daily life. Throughout American history these issues have ranged from slavery to social issues, and from civil rights to regional issues.

One additional thing this project will explore is which ideological dimension was the most important in the prediction of the votes. Are the machine learning models more impacted by the first or second dimension? By the Nominate or Nokken-Poole estimates?

Since the possible outcomes are votes expressed in favor or against a legislation (yea or nea), this is a binary classification problem and will be treated as such throughout this analysis.

Data and Variables

Two datasets are gathered before being joined together (Submitted with the project).

- The voting data for the Civil Rights Act
- The member ideology for the 88th Congress

Voteview also takes other types of votes into account, paired votes, announced yea or nea votes, present and abstention votes. These will not be taken into account in this project. Paired votes are informal arrangements and do not count towards the final official vote count, the same is applicable to announced yea or nea votes. Present and abstention votes are part of the official record, however, predicting these types of votes is not the goal of this project and they have minimal prevalence in the datasets. For these reasons, these votes will simply be droppped.

Method Overview

6 machine learning models will be used:

- Naive Bayes
- Generalized Linear Model (GLM)
- K Nearest Neighbor (KNN)
- Classification and Regression Tree
- Random Forest
- Support Vector Machines (SVM)

Script structure

Though very long, the script doesn't take much time to run in RStudio. If executed, there will be a lot of elements in the environment, the intention is to have a clear overview as every element in the environment has the same name structure to easily diffentiate every element from the other. Every machine learning model's R script has the same structure, after the split into a training and test set, and is explained here.

1. A seed is set to assure the reproducibility of the results.
2. 10 fold cross-validation is performed with the trainControl function.
3. The model is trained on the training set using the train function.
4. Prediction object is created for the AUC-PR evaluation.
5. Creation of the model's confusionmatrix.
6. Extract accuracy, recall, specificity, F1-score and AUC-PR before being put in a table.

As one would expect, the code is quite repetitive. The golden rule would be to write a function to avoid rewriting the same line of code over and over again. However, given the values needed to determine certain outputs and results, this would take considerable time and such useful yet exaustive manipulations are beyond my ability at the time of writing this.

Evaluation

The results are shown at the end in tables where all the models are compared with eachother. This is to have a concise overview of the different algorithms that were used.

Evaluation will be based on three parameters: the overall accuracy displayed with specificity, recall and precision of the model, the F1-Score, and AUC-PR.

Why three different evaluations? Because each evaluation method gives different insights and evaluates the model differently.

Accuracy, Recall, Specifity, Precision

Accuracy, specificity and recall is usually calculated via a confusion matrix, which looks like the table below. Briefly summarized, A confusion matrix is a technique for summarizing the performance of a classification algorithm.

	Actual	
Prediction	yea	nea
yea	True Positive (TP)	False Positive (FP)
nea	False Negative (FN)	True Negative (TN)

Here's what the values mean in the context of this project.

- **True Positive (TP)**
  - The model predicted "yea" which is the same as the actual data.
- **True Negative (TN)**
  - The model predicted "nea" which is the same as the actual data.
- **False Positive (FP)**
  - The model predicted "yea" when it was actually "nea".
- **False Negative (FN)**
  - The model predicted "nea" when it was actually "yea".

Accuracy is the proportion of predictions that were predicted correctly.

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

Specificity is the proportion of actual negative predictions (nea votes) that were predicted correctly as negative.

$$Specificity = \frac{TN}{TN + FP}$$

Recall is the proportion of actual positive predictions (yea votes) that were predicted correctly as positive. Recall refers to the percentage of total relevant results correctly classified by the model.

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

Precision gives information about a model's performance regarding the false positives. Precision refers to the fraction of relevant instances among the total retrieved instances (in this case yea votes). It's important to note that unlike specificity and recall, precision depends on prevalence since higher prevalence implies that you can get higher precision even when guessing.

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

In machine learning, there is a trade-off between precision and recall. Examplified by this project, a model with high precision (most of the votes found by the model were yea votes), will have low recall (a lot of yea votes were missed by the model). On the other hand, a model with high recall (a lot of the yea votes were found) will have low precision (the model also predicted a lot fo nea votes). To find the best possible trade-off, the F1 score is used.

F1-Score

The F1-score is an evaluation method that keeps precision and recall in check. in other words it finds the best possible combination of recall and precision. It's mainly used when classes are not-evenly distributed, it can be argued that this is the case here for this project, since there are more “yea” votes than “nea” votes (70-30%).

The F1-Score is calculated this way.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

AUC-PR

AUC-PR stands for Area under the Precision-Recall curve. A PR curve is a graph with Precision values on the y-axis and Recall values on the x-axis. It plots the precision of a model against its recall, and represents every possible trade-off between precision and recall. If the Area Under the Curve for a model is high then that model has high precision and high recall. This is a key evaluation method for the models of this project since prevalence matters in this examination, there are more yea votes than nea votes.

Method & Analysis

Load Libraries

```
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(caret)) install.packages("caret")
if(!require(readxl)) install.packages("readxl")
if(!require(gridExtra)) install.packages("gridExtra")
if(!require(grid)) install.packages("grid")
if(!require(rpart.plot)) install.packages("rpart.plot")
if(!require(PRROC)) install.packages("PRROC")
if(!require(kableExtra)) install.packages("kableExtra") #make the tables in rmarkdown
if(!require(float)) install.packages("float") #for table formating in rmarkdown
```

Data Importation & First Look

Files must be in working directory

```
house_cra_votes <- read_excel("20200512_0813_voteview_download.xls", sheet = 1)
h88 <- read_csv("H088_members.csv")

head(house_cra_votes,5)
```

```
## # A tibble: 5 x 5
##   icpsr id         name                          state_abbrev    V1
##   <dbl> <chr>      <chr>                         <chr>        <dbl>
## 1     2 MH08800002 ABBITT, Watkins Moorman       VA               6
## 2 10249 MH08810249 WILSON, Robert Carlton (Bob)  CA               1
## 3    13 MH08800013 ABERNETHY, Thomas Gerstle     MS               6
## 4  4110 MH08804110 HARRIS, Oren                  AR               6
## 5  2065 MH08802065 CORBETT, Robert James         PA               1
```

```
head(h88, 5)
```

```
## # A tibble: 5 x 22
##   congress chamber icpsr state_icpsr district_code state_abbrev party_code
##      <dbl> <chr>   <dbl>       <dbl>         <dbl> <chr>             <dbl>
## 1       88 Presid~ 99902          99             0 USA                 100
## 2       88 Presid~ 99903          99             0 USA                 100
## 3       88 House     195          41            98 AL                  100
## 4       88 House    2909          41            98 AL                  100
## 5       88 House    3754          41            98 AL                  100
## # ... with 15 more variables: occupancy <dbl>, last_means <dbl>, bioname <chr>,
## #   bioguide_id <chr>, born <dbl>, died <dbl>, nominate_dim1 <dbl>,
## #   nominate_dim2 <dbl>, nominate_log_likelihood <dbl>,
## #   nominate_geo_mean_probability <dbl>, nominate_number_of_votes <dbl>,
## #   nominate_number_of_errors <dbl>, conditional <lgl>,
## #   nokken_poole_dim1 <dbl>, nokken_poole_dim2 <dbl>
```

```r
str(data.frame(house_cra_votes), width = 80, strict.width = "cut")
```

```
## 'data.frame':    432 obs. of  5 variables:
##  $ icpsr       : num  2 10249 13 4110 2065 ...
##  $ id          : chr  "MH08800002" "MH08810249" "MH08800013" "MH08804110" ...
##  $ name        : chr  "ABBITT, Watkins Moorman" "WILSON, Robert Carlton (Bob)"..
##  $ state_abbrev: chr  "VA" "CA" "MS" "AR" ...
##  $ V1          : num  6 1 6 6 1 1 6 6 1 1 ...
```

```r
str(data.frame(h88), width = 80, strict.width = "cut")
```

```
## 'data.frame':    445 obs. of  22 variables:
##  $ congress                     : num  88 88 88 88 88 88 88 88 88 88 ...
##  $ chamber                      : chr  "President" "President" "House" "House"..
##  $ icpsr                        : num  99902 99903 195 2909 3754 ...
##  $ state_icpsr                  : num  99 99 41 41 41 41 41 41 41 41 ...
##  $ district_code                : num  0 0 98 98 98 98 98 98 98 98 ...
##  $ state_abbrev                 : chr  "USA" "USA" "AL" "AL" ...
##  $ party_code                   : num  100 100 100 100 100 100 100 100 100 100..
##  $ occupancy                    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ last_means                   : num  0 0 1 1 1 1 1 1 1 1 ...
##  $ bioname                      : chr  "KENNEDY, John Fitzgerald" "JOHNSON, L"..
##  $ bioguide_id                  : chr  "K000107" "J000160" "A000206" "E000120"..
##  $ born                         : num  1917 1908 1906 1913 1897 ...
##  $ died                         : num  1963 1973 1971 1999 1982 ...
##  $ nominate_dim1                : num  -0.472 -0.337 -0.03 -0.381 -0.117 -0.11..
##  $ nominate_dim2                : num  -0.487 -0.035 1 0.905 0.993 0.833 0.866..
##  $ nominate_log_likelihood      : num  -14.9 -25.5 -66.8 -18.7 -59.2 ...
##  $ nominate_geo_mean_probability: num  0.919 0.897 0.695 0.891 0.669 0.615 0.8..
##  $ nominate_number_of_votes     : num  176 236 184 161 147 186 137 115 147 173..
##  $ nominate_number_of_errors    : num  4 7 35 6 34 52 9 4 24 37 ...
##  $ conditional                  : logi  NA NA NA NA NA NA ...
##  $ nokken_poole_dim1            : num  NA NA 0.044 -0.421 0.028 0.013 -0.348 -..
##  $ nokken_poole_dim2            : num  NA NA 0.999 0.907 0.947 0.881 0.937 0.9..
```

The values of all the variables are explained on the voteview website.

The variables needed are the following:

- nominate_dim1
- nominate_dim2
- nokken_poole_dim1
- nokken_poole_dim2
- V1 (Which represents the votes on a 1 to 6 scale)

Data Cleaning and Preparation

The two data frames are first joined together to form a single dataset.

```
cra_data <- inner_join(h88, house_cra_votes)
cra_data <- as.data.frame(cra_data)
```

Not needed variables are removed and the V1 column with the votes is renamed.

```
cra_data <- cra_data %>%
  mutate(party = if_else(party_code == "100", "democrat", "republican")) %>%
  filter(str_detect(chamber, "President", negate = T)) %>%
  rename(vote = V1) %>%
  select(nominate_dim1, nominate_dim2, nokken_poole_dim1, nokken_poole_dim2, party, vote)
```

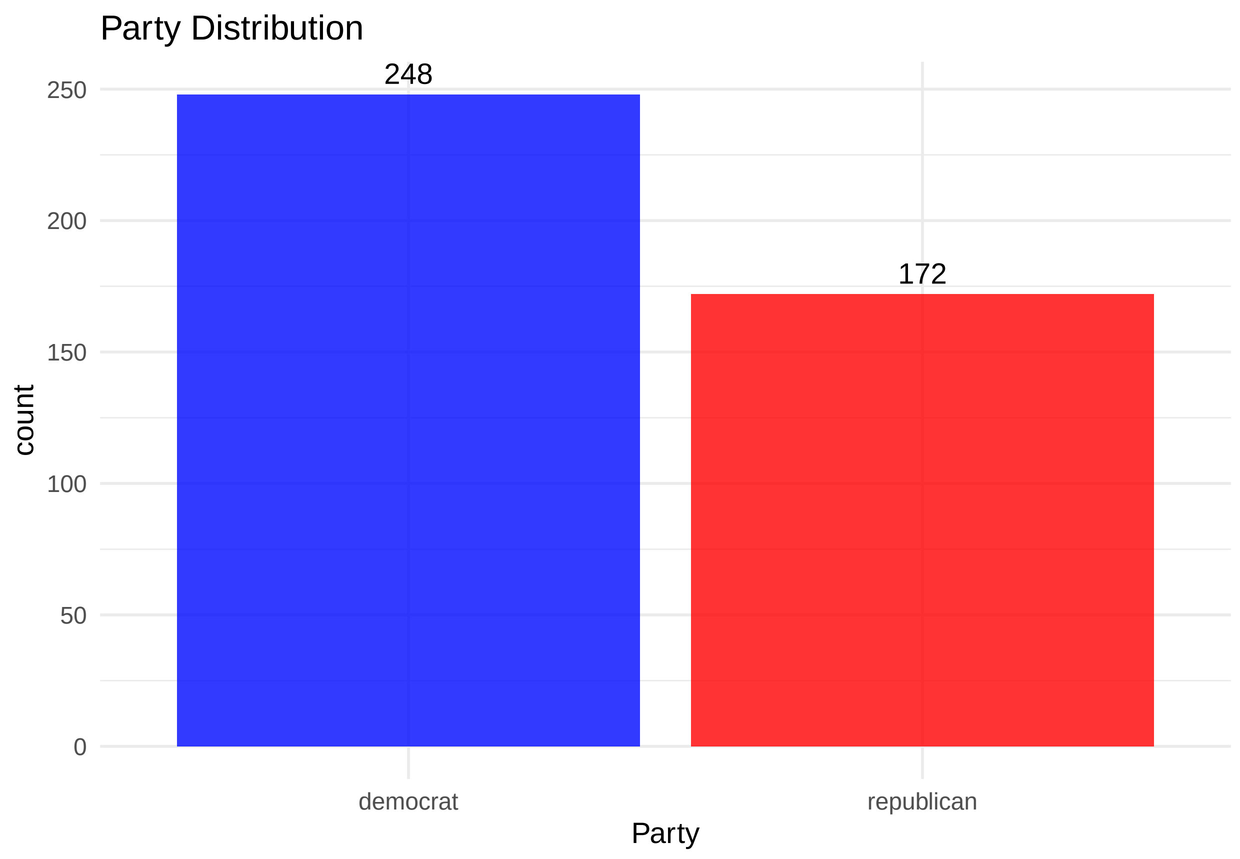
To make everything more digest, the values of the votes are changed to "yea" and "nea" factor levels. All other values are dropped as explained above.

```
not_voted <- c(2,3,5,7)
cra_data <- cra_data %>%
  mutate(vote = replace(vote, vote == 1, "yea")) %>%
  mutate(vote = replace(vote, vote == 6, "nea")) %>%
  filter(!vote %in% not_voted) %>%
  mutate(vote = as.factor(vote)) %>%
  mutate(party = as.factor(party))
```

Data Visualization

Party Distribution

The Democrats retained control of the House of Representatives after the 1962 elections, as illustrated by this plot. Although this doesn't make a lot of difference, keep in mind that certain party-members were dropped because of their vote.



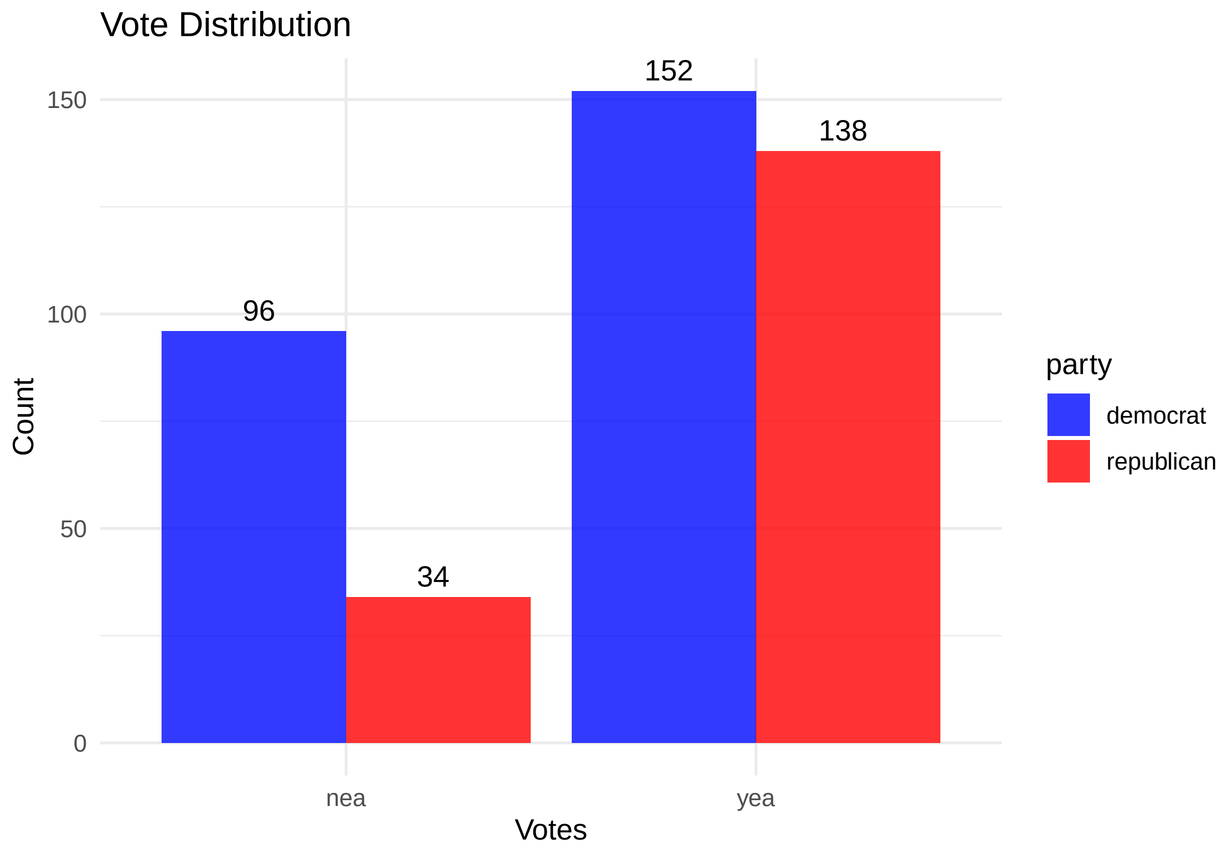
Vote Distribution

Table 1: Votes

Vote	Freq
nea	130
yea	290

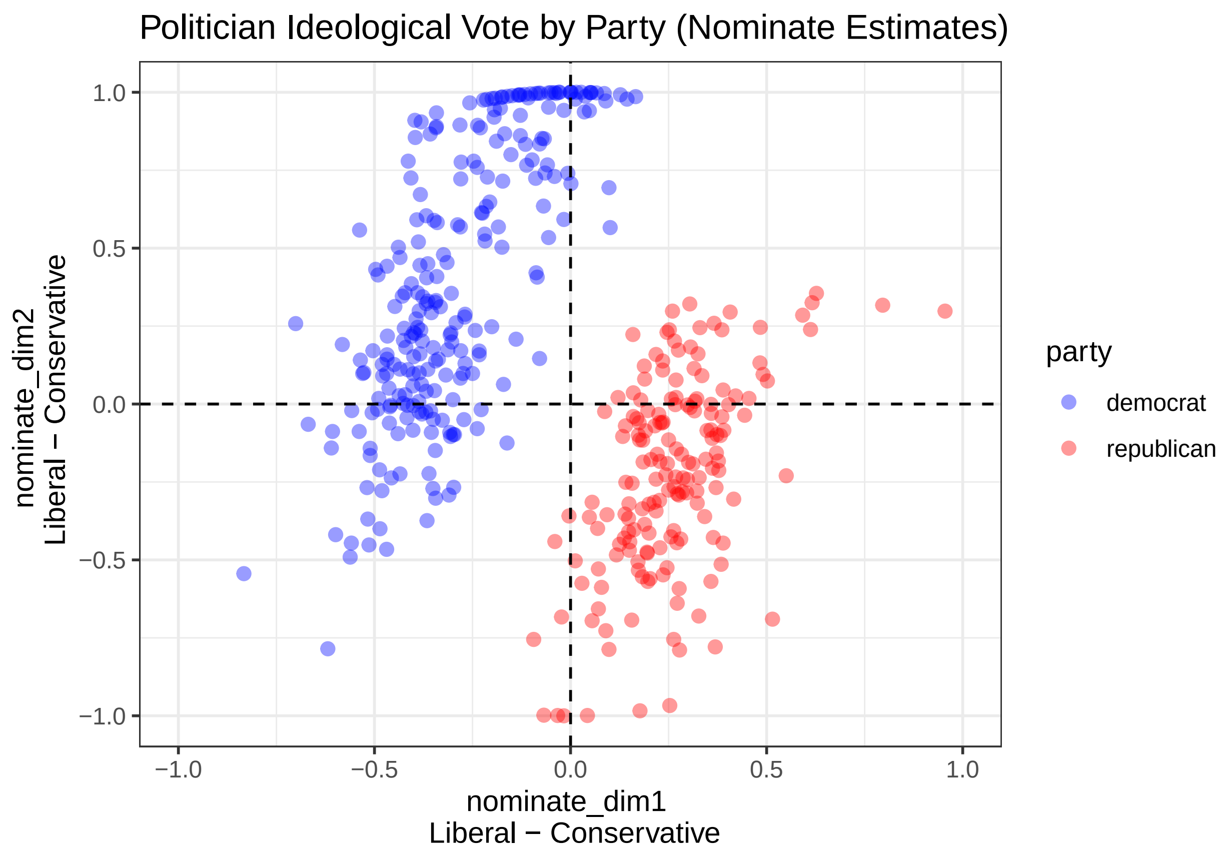
As one can see, there are more yea votes than nea votes, but how are they distributed?

The plot below shows how every party member voted.



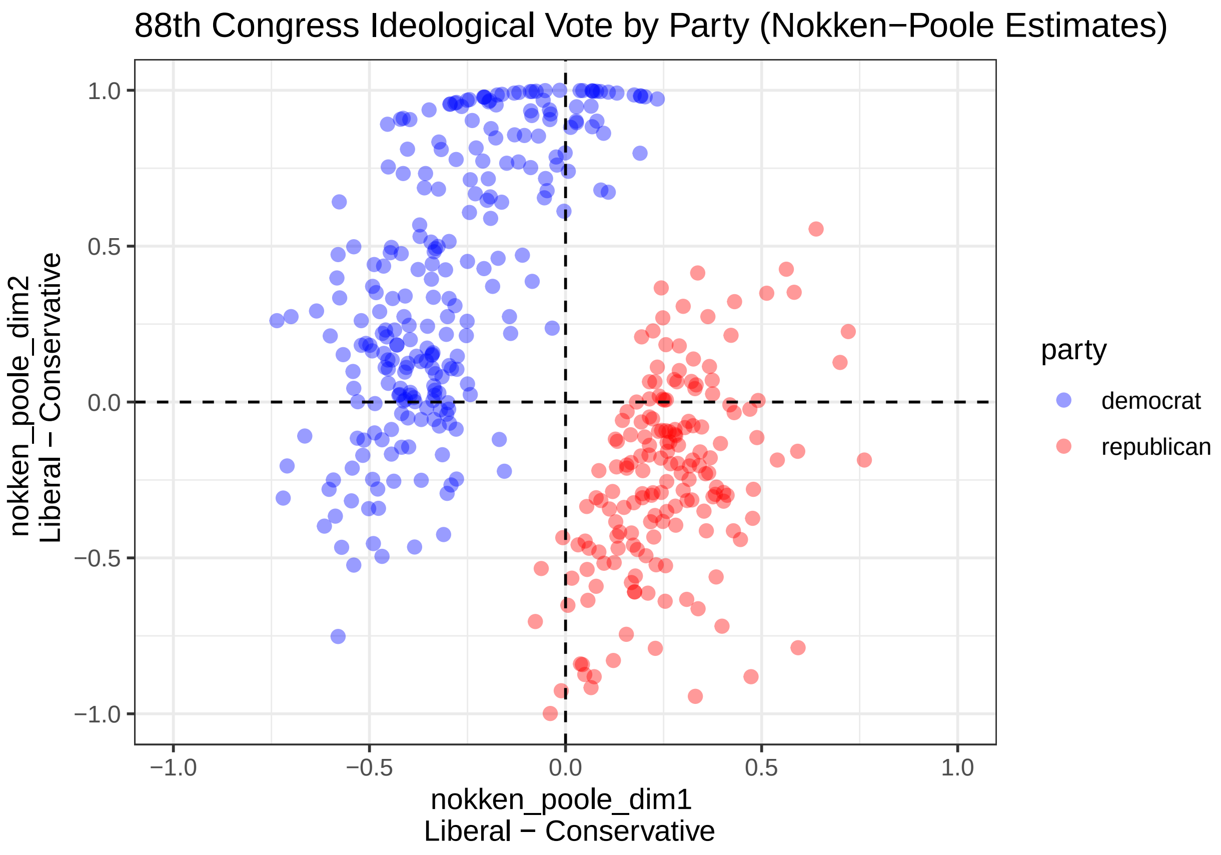
Keep in mind that in the 1960s the Democratic party was very popular in the South and held on to different values from those that were defended by their colleagues in other regions of the US. The plot below represents the ideology off all the members of the house throughout their career. This is based on the nominate parameter.

Nominate Estimates



As shown above, Democrats are more inclined to find themselves on the side of economic liberalism whereas Republicans are more on the conservative side. However, for the second dimension, when it comes to slavery, currency, nativism, civil rights, and lifestyle issues democrats tend to be more on the conservative side. The same applies to the Nokken-Poole Estimates (see below) which assumes that each congress is completely separate for the purposes of estimating a member's ideology.

Nokken-Poole Estimates



One notices that the plots are similar to the one found on the vote's rollcall webpage on voteview

Models

Training and Test Set

```r
set.seed(1997, sample.kind = "Rounding")

test_index <- createDataPartition(cra_data$vote, times = 1, p= 0.20, list = F)

train_cra <- cra_data[-test_index,]

test_cra <- cra_data[test_index,]
```

Nominate Estimates

Naive Bayes

```r
set.seed(13, sample.kind = "Rounding")
ctrl <- trainControl(method="cv", number = 10, p = 0.9, savePredictions = T, classProbs = T)

nb <- train(vote ~ nominate_dim1 + nominate_dim2 + nokken_poole_dim1 + nokken_poole_dim2,
                data = train_cra, method = "nb", trControl = ctrl)

pred_nb <- predict(nb, test_cra, type = "prob")

cm_nb <- confusionMatrix(predict(nb, test_cra), test_cra$vote, positive = "yea")

results <- tibble(model = "Naive Bayes",
                      accuracy = cm_nb$overall["Accuracy"],
                      recall = cm_nb$byClass["Sensitivity"],
                      specificity = cm_nb$byClass["Specificity"],
                      precision = cm_nb$byClass["Pos Pred Value"],
                      f1_score = F_meas(predict(nb, test_cra), test_cra$vote),
                      AUC_PR = pr.curve(scores.class0 = pred_nb$yea[test_cra$vote == "yea"],
                                        scores.class1 = pred_nb$nea[test_cra$vote == "nea"],
                                        curve = T)$auc.integral)
```

Table 2: Naive Bayes

model	accuracy	recall	specificity	precision	f1_score	AUC_PR
Naive Bayes	0.869	0.983	0.615	0.851	0.744	0.605

Generalized Linear Model (GLM)

```
set.seed(73, sample.kind = "Rounding")

ctrl <- trainControl(method="cv", number = 10, p = 0.9,
                     savePredictions = T, classProbs = T)

glm <- train(vote ~ nominate_dim1 + nominate_dim2 + nokken_poole_dim1 + nokken_poole_dim2,
                 data = train_cra, method = "glm", trControl = ctrl)

pred_glm <- predict(glm, test_cra, type = "prob")

cm_glm <- confusionMatrix(predict(glm, test_cra), test_cra$vote, positive = "yea")

results <- results %>%
  add_row(model = "GLM",
          accuracy = cm_glm$overall["Accuracy"],
          specificity = cm_glm$byClass["Specificity"],
          recall = cm_glm$byClass["Sensitivity"],
          precision = cm_glm$byClass["Pos Pred Value"],
          f1_score = F_meas(predict(glm, test_cra), test_cra$vote),
          AUC_PR = pr.curve(scores.class0 = pred_glm$yea[test_cra$vote == "yea"],
                            scores.class1 = pred_glm$nea[test_cra$vote == "nea"],
                            curve = T)$auc.integral)
```

Table 3: Generalized Linear Model

model	accuracy	recall	specificity	precision	f1_score	AUC_PR
GLM	0.94	0.983	0.846	0.934	0.898	0.819

K Nearest Neighbors (KNN)

When an algorithm includes a tuning parameter like KNN (number of neighbors), train() automatically uses cross validation to decide amongst a few default values and pick the one with the highest accuracy. For knn the default is to try k = 5, 7, 9. However, here 50 values between 1 and 50 are tried out.

```r
set.seed(23, sample.kind = "Rounding")

ctrl <- trainControl(method="cv", number = 10, p = 0.9,
                     savePredictions = T, classProbs = T)

knn <- train(vote ~ nominate_dim1 + nominate_dim2 + nokken_poole_dim1 + nokken_poole_dim2,
             data = train_cra, method = "knn",
             tuneGrid = data.frame(k = seq(1,50)), trControl = ctrl)
knn$bestTune
```

```
##     k
## 13 13
```

The parameter that maximizes the accuracy is k = 13. This gives the following results

```r
pred_knn <- predict(knn, test_cra, type = "prob")

cm_knn <- confusionMatrix(predict(knn, test_cra),test_cra$vote, positive = "yea")

results <- results %>%
  add_row(model = "KNN",
          accuracy = cm_knn$overall["Accuracy"],
          specificity = cm_knn$byClass["Specificity"],
          recall = cm_knn$byClass["Sensitivity"],
          precision = cm_knn$byClass["Pos Pred Value"],
          f1_score = F_meas(predict(knn, test_cra), test_cra$vote),
          AUC_PR = pr.curve(scores.class0 = pred_knn$yea[test_cra$vote == "yea"],
                            scores.class1 = pred_knn$nea[test_cra$vote == "nea"],
                            curve = T)$auc.integral)
```

Table 4: K-Nearest Neighbors

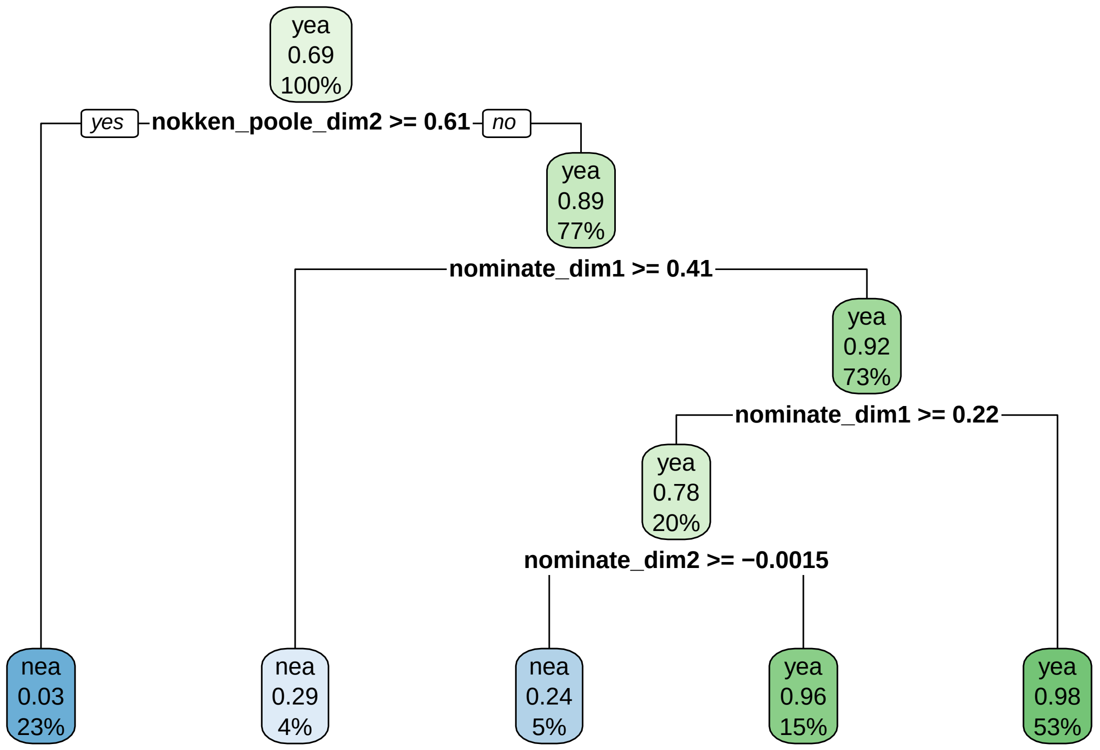
model	accuracy	recall	specificity	precision	f1_score	AUC_PR
KNN	0.929	0.966	0.846	0.933	0.88	0.732

Classification Trees

```r
set.seed(53, sample.kind = "Rounding")

ctrl <- trainControl(method="cv", number = 10, p = 0.9,
                     savePredictions = T, classProbs = T)

tree <- train(vote ~ nominate_dim1 + nominate_dim2 + nokken_poole_dim1 + nokken_poole_dim2,
              data = train_cra, method = "rpart",
              tuneGrid = data.frame(cp = seq(0, 0.05, len = 25)), trControl = ctrl)

tree$bestTune
```

```
##             cp
## 6 0.01041667
```

The complexity parameter with the highest accuracy is 0.01041667, the decision tree looks like this after the model has trained on the training set. Each node shows the predicted class, probability of being a yea vote and tge percentage of observations in the node.



Here are the results when applied on the test set.

```r
pred_tree <- predict(tree, test_cra, type = "prob")

cm_tree <- confusionMatrix(predict(tree, test_cra), test_cra$vote, positive = "yea")
```

```r
results <- results %>%
  add_row(model = "Class. Tree",
          accuracy = cm_tree$overall["Accuracy"],
          specificity = cm_tree$byClass["Specificity"],
          recall = cm_tree$byClass["Sensitivity"],
          precision = cm_tree$byClass["Pos Pred Value"],
          f1_score = F_meas(predict(tree, test_cra), test_cra$vote),
          AUC_PR = pr.curve(scores.class0 = pred_tree$yea[test_cra$vote == "yea"],
                            scores.class1 = pred_tree$nea[test_cra$vote == "nea"],
                            curve = T)$auc.integral)
```

Table 5: Classification Tree

model	accuracy	recall	specificity	precision	f1_score	AUC_PR
Class. Tree	0.917	0.948	0.846	0.932	0.863	0.925

Random Forest

```r
set.seed(42, sample.kind = "Rounding")

ctrl <- trainControl(method="cv", number = 10, p = 0.9,
                     savePredictions = T, classProbs = T)

rf <- train(vote ~ nominate_dim1 + nominate_dim2 + nokken_poole_dim1 + nokken_poole_dim2,
            importance = T, data = train_cra, method = "rf",
            tuneGrid = data.frame(mtry = seq(1,2)), ntree = 100, trControl = ctrl)

pred_rf <- predict(rf, test_cra, type = "prob")

cm_rf <- confusionMatrix(predict(rf, test_cra), test_cra$vote, positive = "yea")

results <- results %>%
  add_row(model = "Random Forest",
          accuracy = cm_rf$overall["Accuracy"],
          specificity = cm_rf$byClass["Specificity"],
          recall = cm_rf$byClass["Sensitivity"],
          precision = cm_rf$byClass["Pos Pred Value"],
          f1_score = F_meas(predict(rf, test_cra), test_cra$vote),
          AUC_PR = pr.curve(scores.class0 = pred_rf$yea[test_cra$vote == "yea"],
                            scores.class1 = pred_rf$nea[test_cra$vote == "nea"],
                            curve = T)$auc.integral)
```

Table 6: Random Forest

model	accuracy	recall	specificity	precision	f1_score	AUC_PR
Random Forest	0.94	0.983	0.846	0.934	0.898	0.734

Support Vector Machines

```
set.seed(117, sample.kind = "Rounding")

ctrl <- trainControl(method="cv", number = 10, p = 0.9,
                     savePredictions = T, classProbs = T)

svm <- train(vote ~ nominate_dim1 + nominate_dim2 + nokken_poole_dim1 + nokken_poole_dim2,
                 data = train_cra, method = "svmLinear", trControl = ctrl,
                 tuneGrid = data.frame(C = seq(1, 5, len = 25)))

svm$bestTune
```

```
##          C
## 2 1.166667
```

The ideal cost parameter is 1.166667, this gives the following results

```
pred_svm <- predict(svm, test_cra, type = "prob")

cm_svm <- confusionMatrix(predict(svm, test_cra), test_cra$vote, positive = "yea")

results <- results %>%
  add_row(model = "SVM",
          accuracy = cm_svm$overall["Accuracy"],
          specificity = cm_svm$byClass["Specificity"],
          recall = cm_svm$byClass["Sensitivity"],
          precision = cm_svm$byClass["Pos Pred Value"],
          f1_score = F_meas(predict(svm, test_cra), test_cra$vote),
          AUC_PR = pr.curve(scores.class0 = pred_svm$yea[test_cra$vote == "yea"],
                            scores.class1 = pred_svm$nea[test_cra$vote == "nea"],
                            curve = T)$auc.integral)
```

Table 7: Support Vector Machines

model	accuracy	recall	specificity	precision	f1_score	AUC_PR
SVM	0.94	0.983	0.846	0.934	0.898	0.839

Results

Accuracy

Table 8: Models Ranked by Accuracy

model	accuracy	recall	precision	specificity
GLM	0.940	0.983	0.934	0.846
Random Forest	0.940	0.983	0.934	0.846
SVM	0.940	0.983	0.934	0.846
KNN	0.929	0.966	0.933	0.846
Class. Tree	0.917	0.948	0.932	0.846
Naive Bayes	0.869	0.983	0.851	0.615

If models had to be chosen based on accuracy, then one would have the choice between GLM, Random Forest, and SVM.

F1- Score

Table 9: Models Ranked by F1-Score

model	f1_score	recall	precision	specificity
GLM	0.898	0.983	0.934	0.846
Random Forest	0.898	0.983	0.934	0.846
SVM	0.898	0.983	0.934	0.846
KNN	0.880	0.966	0.933	0.846
Class. Tree	0.863	0.948	0.932	0.846
Naive Bayes	0.744	0.983	0.851	0.615

As expected given the recall, and precision results which are also the same, the F1-score also exactly the same for GLM, Random Forest and SVM.

Why is that?

First of all, accuracy is influenced by the prevalence or imbalance of the dataset, in this case there are 70% yea votes for 30% nea votes. So in the case of this dataset, accuracy would not be a right evaluation method.

Second, although the F1 score takes imbalance into account, the default classification threshold value for all models in the caret library is set to 0.5. In simple terms the threshold or cut-off represents, in the case of this project, the probability that the prediction made by the model is true (a yea vote). It's a representation of the trade-off between false positives and false negatives.

Because the caret package sets this by default to 0.5, the results are not optimal. the tables below display the accuracy and f1 values for every possible threshold point between 0.1 and 1 for the GLM and SVM method.

Table 10: GLM Accuracy for every threshold value

parameter	**prob_threshold**	**Accuracy**
none	0.0	0.3094143
none	0.1	0.8841940
none	0.2	0.9196817
none	0.3	0.9284212
none	**0.4**	**0.9489305**
none	**0.5**	**0.9432213**
none	0.6	0.9283321
none	0.7	0.9253909
none	0.8	0.9107690
none	0.9	0.9137102
none	1.0	0.6905857

Table 11: SVM Accuracy for every threshold value

C	**prob_threshold**	**Accuracy**
1.166667	0.0	0.3093812
1.166667	0.1	0.8659919
1.166667	0.2	0.9196664
1.166667	0.3	0.9228699
1.166667	**0.4**	**0.9463305**
1.166667	**0.5**	**0.9434785**
1.166667	0.6	0.9343876
1.166667	0.7	0.9227858
1.166667	0.8	0.9199287
1.166667	0.9	0.8957754
1.166667	1.0	0.6906188

For accuracy, when the treshold is set at 0.5, the value is the same for both models. Additionally, 0.5 isn't even the threshold with the highest accuracy, it's 0.4 for both models. At a 0.4 threshold the GLM performs slightly better than SVM.

Table 12: GLM F1-Score for every threshold value

parameter	**prob_threshold**	**F1**
none	0.0	0.4725054
none	0.1	0.8412177
none	0.2	0.8836942
none	0.3	0.8930973
none	**0.4**	**0.9204863**
none	**0.5**	**0.9073236**
none	0.6	0.8788358
none	0.7	0.8711416
none	0.8	0.8392506
none	0.9	0.8372050
none	1.0	NaN

Table 13: SVM F1-Score for every threshold value

C	**prob_threshold**	**F1**
1.166667	0.0	0.4724997
1.166667	0.1	0.8188228
1.166667	0.2	0.8796010
1.166667	0.3	0.8808529
1.166667	**0.4**	**0.9098950**
1.166667	**0.5**	**0.8978249**
1.166667	0.6	0.8802613
1.166667	0.7	0.8582662
1.166667	0.8	0.8480574
1.166667	0.9	0.7924407
1.166667	1.0	NaN

For the F1-score, when the treshold is set at 0.5, the value is the very close to the same for both models. Like with GLM, 0.5 isn't even the threshold with the highest accuracy, it's 0.4 for both models. and just like GLM at a 0.4 threshold the GLM performs better than SVM.

There is a way to change the thresholds in the caret package, however, it is beyond the scope of this course and beyond my own understanding and skills in R-programming

AUC-PR

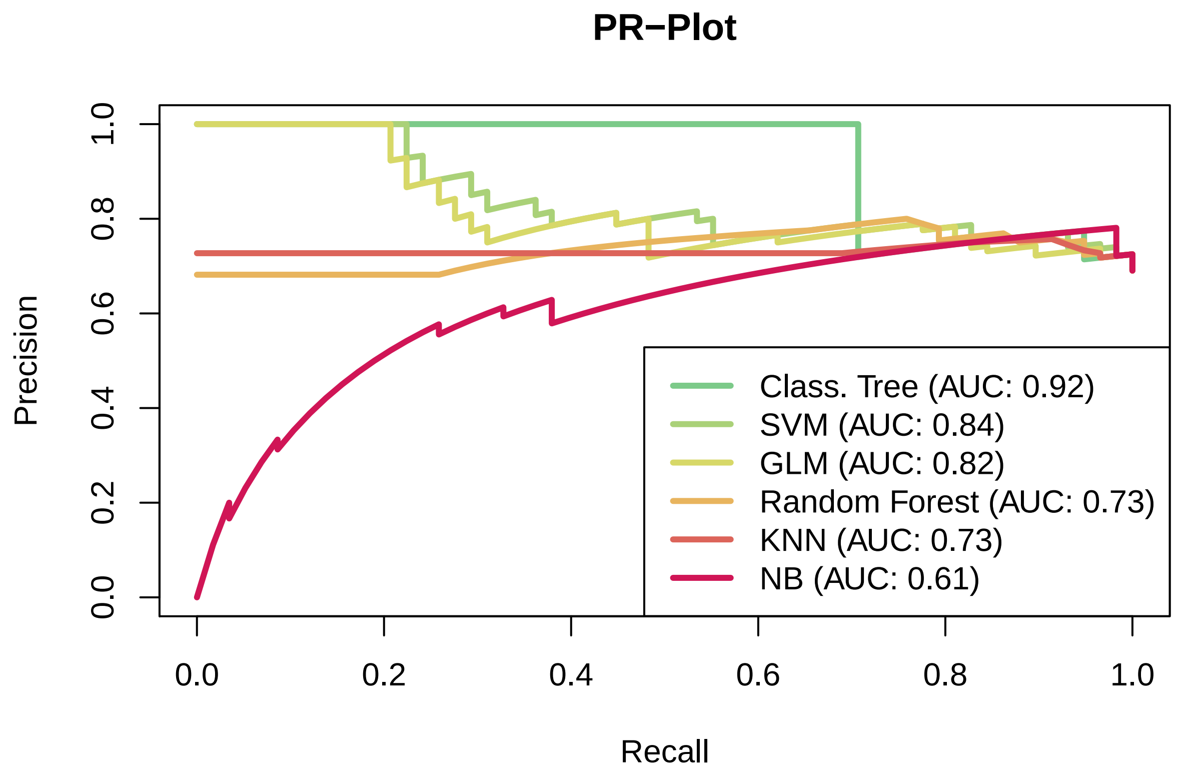
What if the models are evaluated based on the AUC-PR which does take prevalence into account and uses different probability thresholds.

Table 14: Models Ranked by AUC-PR Score

model	**AUC_PR**
Class. Tree	0.925
SVM	0.839
GLM	0.819
Random Forest	0.734
KNN	0.732
Naive Bayes	0.605

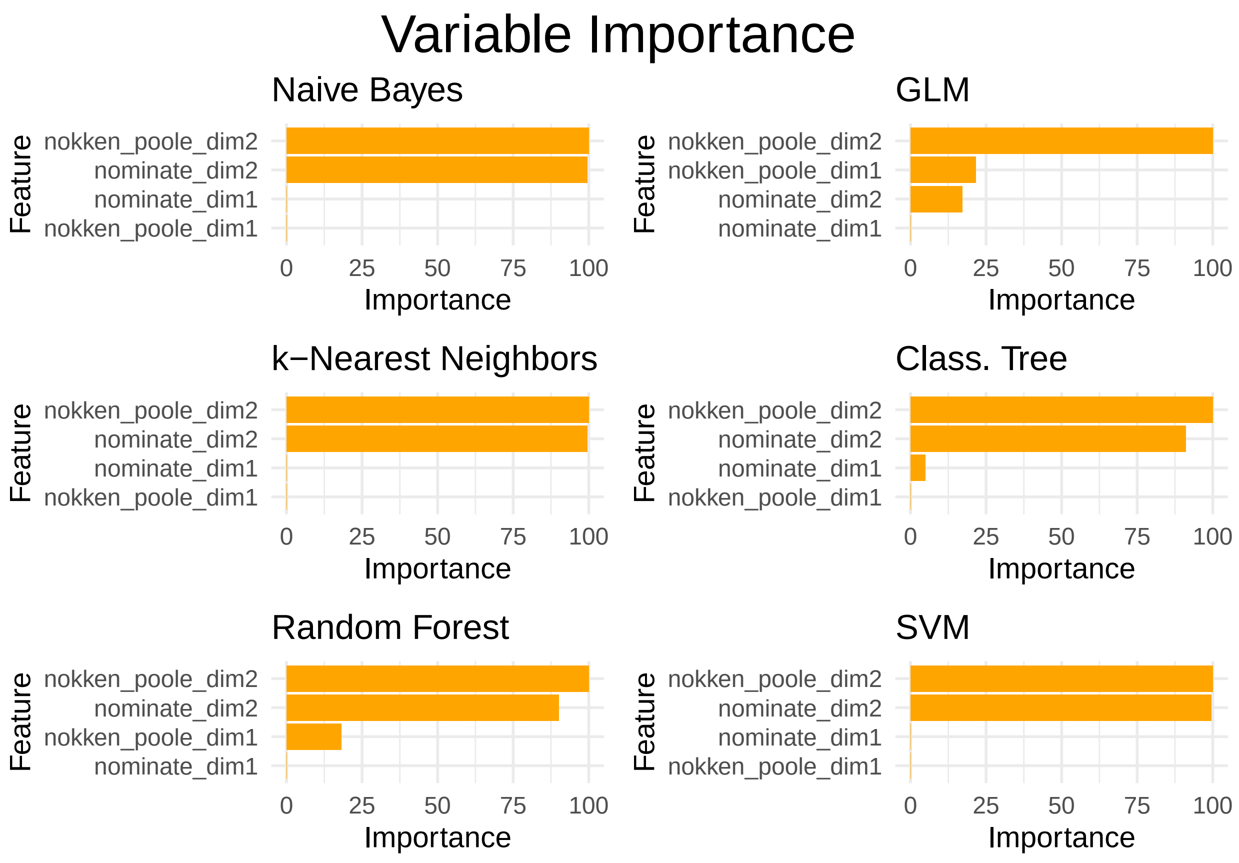
There is a considerable difference between the AUC values of the models. The Classification Tree model performs the best out of all the models by a considerable margin.

For a visual perspective the PR plots are shown below.



Predictor Importance

What was the predictor that had the highest impact on the models? Which predictor has the biggest impact on the classification of the votes when that predictor is modified? The plot below has the answer



For all the models the nokken_poole_dim2 predictor is the most influential one. This suggests that the prediction of the vote is more affected when every congress is taken separately with the second dimension, which represents day to day issues. This implies that members of the House of Representatives did not vote to what they originally believed in, but set aside their predisposed beliefs to determine the outcome of such a critical vote in US history.

Conclusion

Given the data and how the models performed, the Classification Tree model would be the most suitable model to predict the votes for the House of Representatives for the Civil Rights act of 1964 (who knows maybe others as well). With randomized data, the accuracy and F1 score is still high compared to the other models. Even with the prevalence of the dataset and without any threshold limitations, as the PR curve shows, the model has a high AUC value which makes it the best-performing model out of the other six.

As an American historian, this project was a blast! It was a tough challege, but oh so rewarding at the end. I'm impatient to discover more R programming skills, and use the techniques that shape the future to better understand the past.

Thank you for taking the time to read through this.

References

- https://voteview.com/about
- https://www.govtrack.us/congress/votes/88-1964/h128
- https://www.govtrack.us/congress/bills/88/hr7152
- Irizarry, Rafael, *Introduction to Data Science (Data Analysis and Prediction Algorithms with R)*, Boca Raton (FL), Taylor & Francis Group (LLC), 2020.
- James, Gareth, Witten, Daniela, Hastie, Trevor, Tibshirani, Robert, *An Introduction to Statistical Learning (With Applications in R),*New-York (NY), Springer, 2013.
- Field, Andy, Field, Zoë, Miles, Jeremy, *Discovering Statistics Using R*, Sage, 2012.