

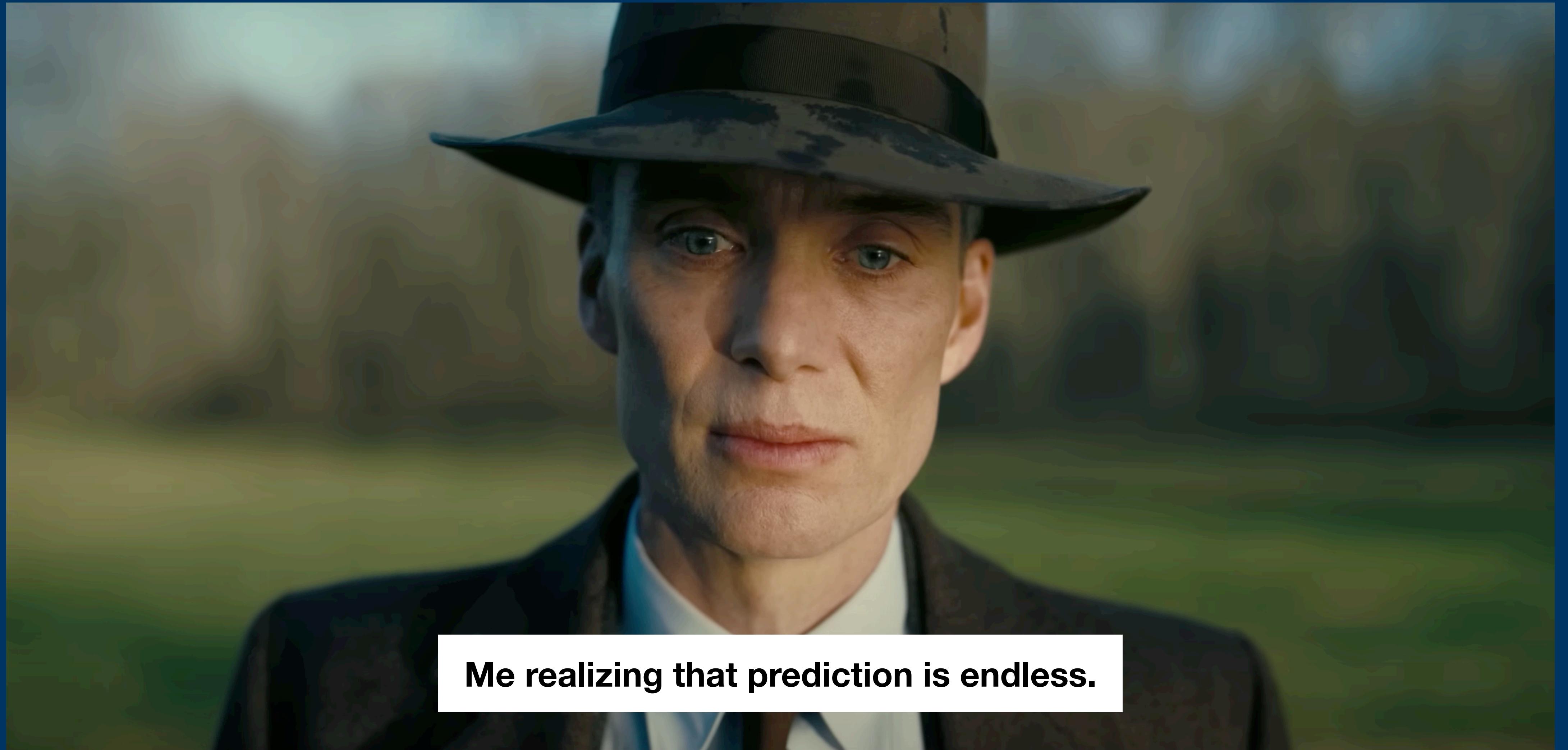
Welcome!

Nameplates please. And technology encouraged today!

All TF materials are available at github.com/nolankav/api-203.

If you want to follow along, download the dataset here:

In R: df <- read.csv("http://tinyurl.com/api-203-tf-5")



Me realizing that prediction is endless.

Prediction, Double Feature R

API 203: TF Session 5

Nolan M. Kavanagh
April 5, 2024

Goals for today

- 1. Briefly review strategies for predicting binary outcomes.**
- 2. Learn how to generate classification models in R.**
- 3. Practice evaluating the accuracy of classification models.**

We'll treat this session like a workshop with an interactive example.

Overview of our sample data

Dataset of ~17,000 U.S. adults surveyed in 2019

casein	Respondent identifier	<i>Cooperative Election Study</i>
commonweight	Post-stratification survey weight	<i>Cooperative Election Study</i>
age	Respondent age (in years)	<i>Cooperative Election Study</i>
gender	Respondent gender	<i>Cooperative Election Study</i>
race_eth	Respondent race/ethnicity	<i>Cooperative Election Study</i>
education	Respondent educational attainment	<i>Cooperative Election Study</i>
marital	Respondent marital status	<i>Cooperative Election Study</i>
employment	Respondent employment status	<i>Cooperative Election Study</i>
device	Device respondent is using for survey	<i>Cooperative Election Study</i>
republican	Identifies as Republican (1) or not (0)	<i>Cooperative Election Study</i>
public_ins	Respondent has public health insurance (1) or not (0)	<i>Cooperative Election Study</i>
public_option	Supports a public option in Medicare (1) or not (0)	<i>Cooperative Election Study</i>

What if we're predicting a binary outcome?

The classic story is that OLS is not appropriate for predicting a binary outcome because you might get predicted values above 1 or below 0.

This is impossible since we can't have a probability above 100% or below 0%.

For this reason, people turn to probit (common in economics) or logit (common everywhere else).

But here's the thing.

All three tend to produce similar predicted probabilities!

And if you have fully interacted categorical predictors,
they produce *identical* predicted probabilities.

So does it matter? Not usually.

Use what's most useful to you.

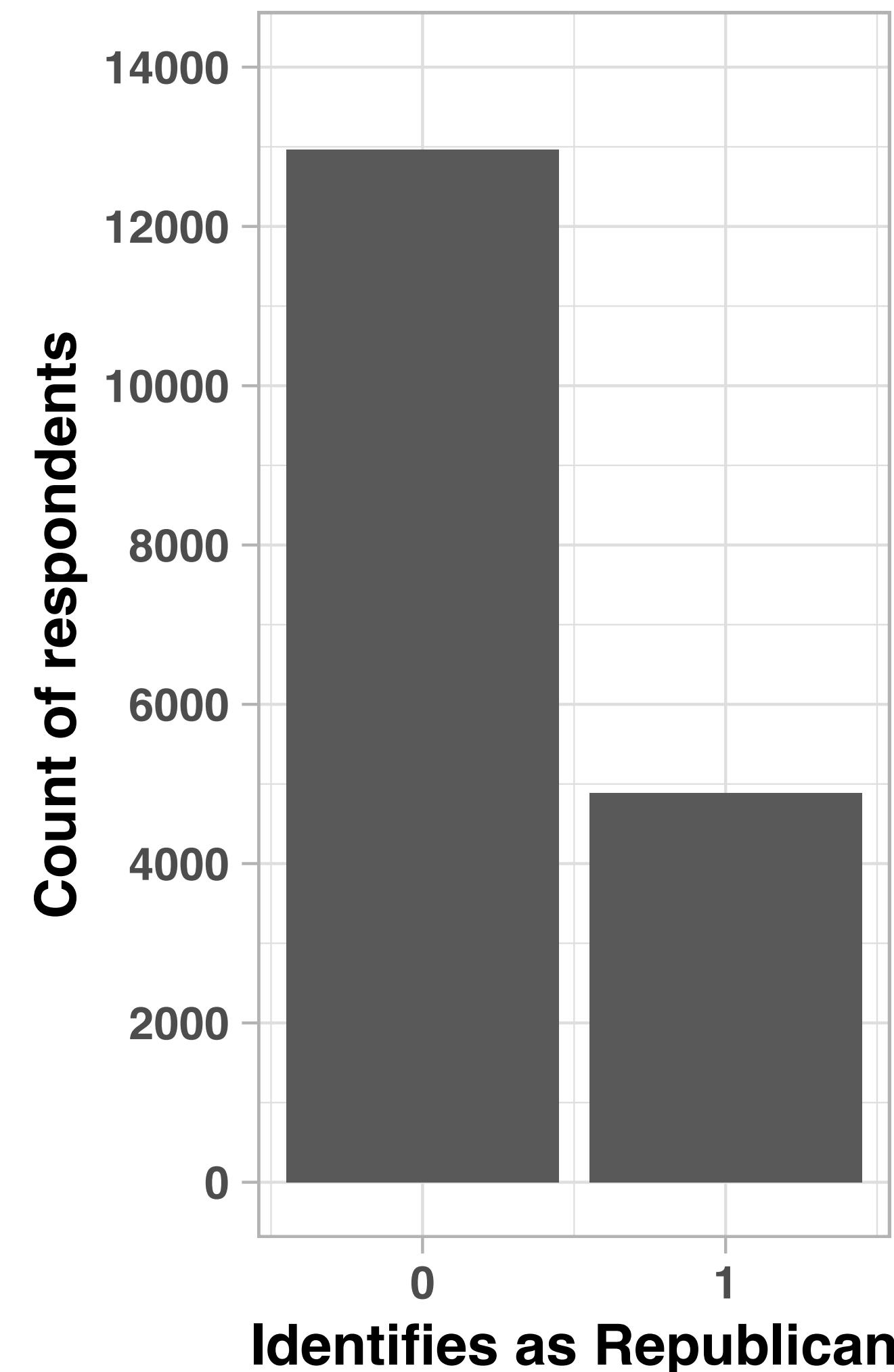


We'll riff on the same idea as last week.

Parties want to know whom they can contact for fundraising, get out the vote, etc.

This time, we will predict if someone is (or isn't) a Republican using the data we have.

```
# Generate bar plot of parties
plot_1 <- ggplot(data=df, aes(x=republican)) +
  geom_bar(stat="count") +
  xlab("Identification as Republican") +
  ylab("Count of respondents") +
  theme_light() +
  theme(text = element_text(size = 10, face = "bold")) +
  scale_x_continuous(breaks = seq(-3, 3, 1)) +
  scale_y_continuous(limits = c(0,14000),
                     breaks = seq(0,15000,2000))
```



Let's try a reasonable model.

```
# Load libraries  
library(pROC)      # ROC tools  
library(rsample)    # Analysis tools
```

This package is great for making ROC curves.
(We'll get to those soon enough.)

```
# Designate training and test sets  
set.seed(1234)  
split    <- initial_split(df, 0.7)  
df_train <- training(split)  
df_test  <- testing(split)
```

Let's go with logit. Set family to binomial.

```
# Logit regression  
model_1 <- glm(republican ~ age + gender + race_eth + education + marital +  
                 employment + device, data=df_train, family="binomial")
```

```
# In-sample prediction  
df_train$predict <- predict(model_1, df_train, type="response")
```

FYI the predict function returns a “latent” value by default. It’s not readily interpretable.

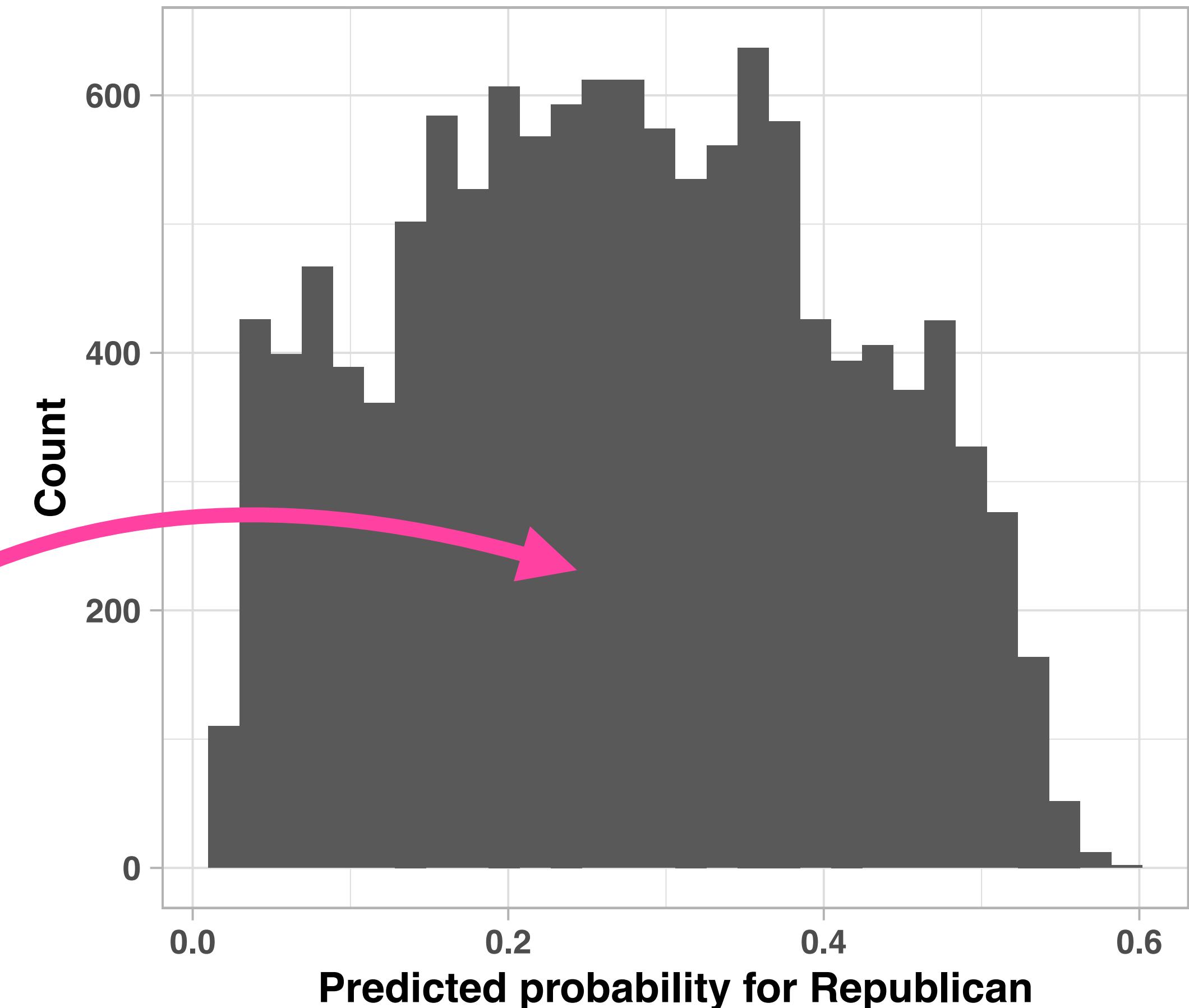
We got actual predicted probabilities by adding type = "response". But it works fine either way.

But here's the problem.

```
# Histogram of predictions
plot_2 <- ggplot(df_train, aes(x=predict)) +
  geom_histogram() +
  theme_light() +
  theme(text = element_text(size = 10, face = "bold")) +
  xlab("Predicted Republican") +
  ylab("Count")
print(plot_2)
```

With classification models, we get continuous predicted values. We have to set a threshold for what we think qualifies as Republican or not.

But which threshold do we pick?



FYI the `predict` function returns a “latent” value by default. It’s not readily interpretable.

We got actual predicted probabilities by adding `type = "response"`. But it works fine either way.

Let's try two different thresholds.

```
# Set potential thresholds
df_train <- df_train %>% mutate(
  threshold_1 = case_when(
    predict > 0.4 ~ 1, TRUE ~ 0),
  threshold_2 = case_when(
    predict > 0.2 ~ 1, TRUE ~ 0
))
```

```
# Evaluate thresholds
table(df_train$republican, df_train$threshold_1)
```

	0	1
0	7782	1315
1	2199	1203

```
table(df_train$republican, df_train$threshold_2)
```

	0	1
0	3640	5457
1	504	2898

Let's try 0.4 and 0.2 as thresholds.



Let's try two different thresholds.

```
# Set potential thresholds
df_train <- df_train %>% mutate(
  threshold_1 = case_when(
    predict > 0.4 ~ 1, TRUE ~ 0),
  threshold_2 = case_when(
    predict > 0.2 ~ 1, TRUE ~ 0
  ))

# Evaluate thresholds
table(df_train$republican, df_train$threshold_1)

  0   1
0 7782 1315
1 2199 1203

table(df_train$republican, df_train$threshold_2)

  0   1
0 3640 5457
1 504 2898
```

		Predicted party	
		Not Rep.	Republican
True party	Not Rep.	True negatives	False positives
	Republican	False negatives	True positives

A brief aside on terminology.

Calculation	Terminology
$FP / (TN + FP)$	False positive rate, 1–Specificity, Type I error
$TP / (TP + FN)$	True positive rate, Recall, Sensitivity, 1–Type II error, Power
TP / All predicted positives	Precision, Positive predicted value
TN / All predicted negatives	Negative predicted value



The poor souls in API 203

“My job is just... dealing with ever so slightly different versions of the exact same statistical concepts”

Let's try two different thresholds.

```
# Set potential thresholds
df_train <- df_train %>% mutate(
  threshold_1 = case_when(
    predict > 0.4 ~ 1, TRUE ~ 0),
  threshold_2 = case_when(
    predict > 0.2 ~ 1, TRUE ~ 0
  ))

# Evaluate thresholds
table(df_train$republican, df_train$threshold_1)

  0   1
0 7782 1315
1 2199 1203

table(df_train$republican, df_train$threshold_2)

  0   1
0 3640 5457
1 504 2898
```

So for 40%:

		Predicted party	
		Not Rep.	Republican
True party	Not Rep.	7782	1315
	Republican	2199	1203

Precision = True positives / All positives =

Recall = True positives / (TP + FN) =

Let's try two different thresholds.

```
# Set potential thresholds
df_train <- df_train %>% mutate(
  threshold_1 = case_when(
    predict > 0.4 ~ 1, TRUE ~ 0),
  threshold_2 = case_when(
    predict > 0.2 ~ 1, TRUE ~ 0
  ))

# Evaluate thresholds
table(df_train$republican, df_train$threshold_1)

      0   1
0 7782 1315
1 2199 1203

table(df_train$republican, df_train$threshold_2)

      0   1
0 3640 5457
1 504 2898
```

So for 40%:

		Predicted party	
		Not Rep.	Republican
True party	Not Rep.	7782	1315
	Republican	2199	1203

Precision = True positives / All positives = $1203 / (1203 + 1315) = 0.48$

Recall = True positives / (TP + FN) = $1203 / (1203 + 2199) = 0.35$

Let's try two different thresholds.

```
# Set potential thresholds
df_train <- df_train %>% mutate(
  threshold_1 = case_when(
    predict > 0.4 ~ 1, TRUE ~ 0),
  threshold_2 = case_when(
    predict > 0.2 ~ 1, TRUE ~ 0
  ))

# Evaluate thresholds
table(df_train$republican, df_train$threshold_1)

      0   1
0 7782 1315
1 2199 1203

table(df_train$republican, df_train$threshold_2)

      0   1
0 3640 5457
1 504 2898
```

So for 20%:

		Predicted party	
		Not Rep.	Republican
True party	Not Rep.	3640	5457
	Republican	504	2898

Precision = True positives / All positives =

Recall = True positives / (TP + FN) =

Let's try two different thresholds.

```
# Set potential thresholds
df_train <- df_train %>% mutate(
  threshold_1 = case_when(
    predict > 0.4 ~ 1, TRUE ~ 0),
  threshold_2 = case_when(
    predict > 0.2 ~ 1, TRUE ~ 0
  ))

# Evaluate thresholds
table(df_train$republican, df_train$threshold_1)

      0   1
0 7782 1315
1 2199 1203

table(df_train$republican, df_train$threshold_2)

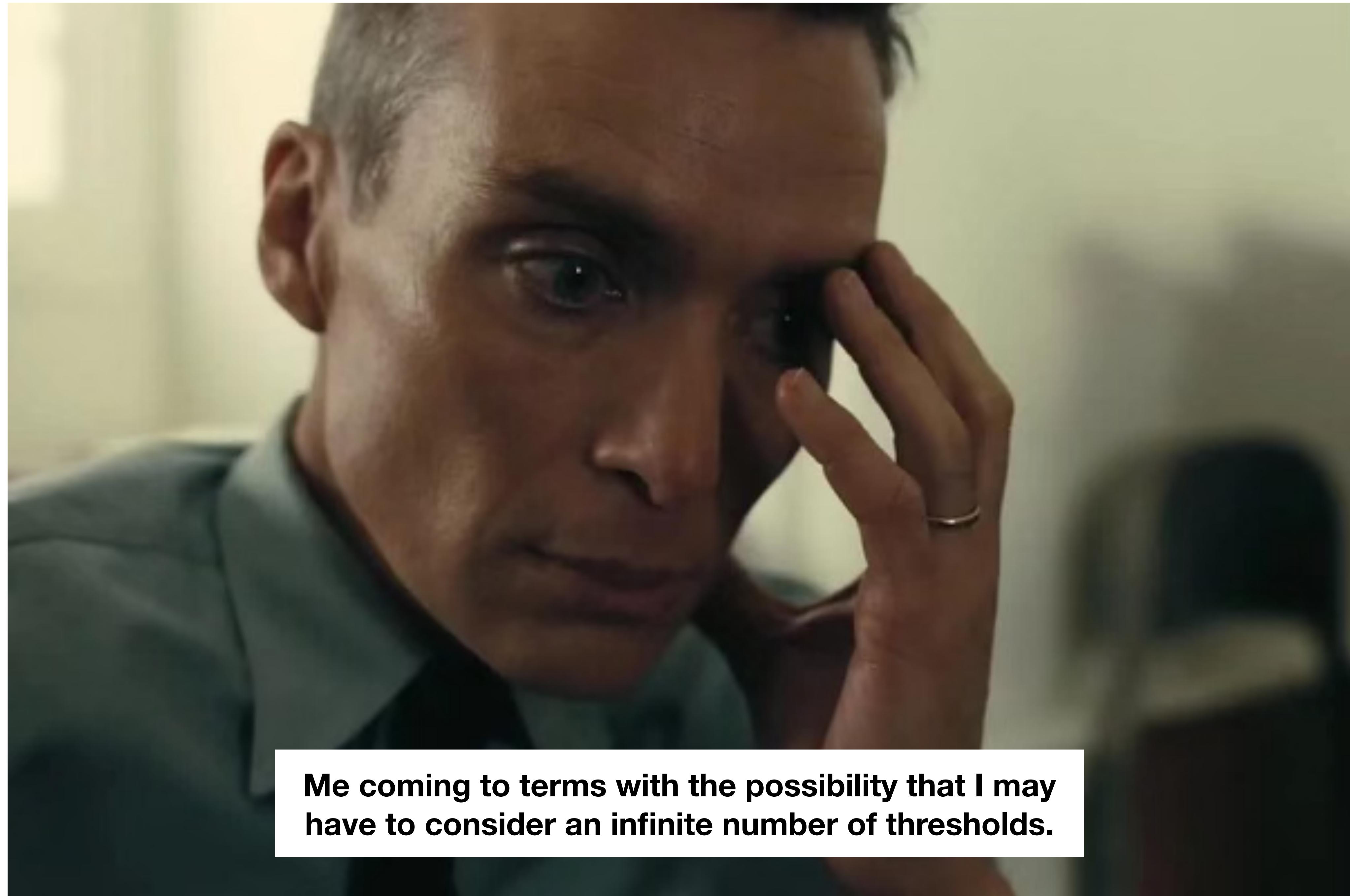
      0   1
0 3640 5457
1 504 2898
```

So for 20%:

		Predicted party	
		Not Rep.	Republican
True party	Not Rep.	3640	5457
	Republican	504	2898

Precision = True positives / All positives = $2898 / (2898 + 5457) = 0.35$

Recall = True positives / (TP + FN) = $2809 / (2898 + 504) = 0.83$

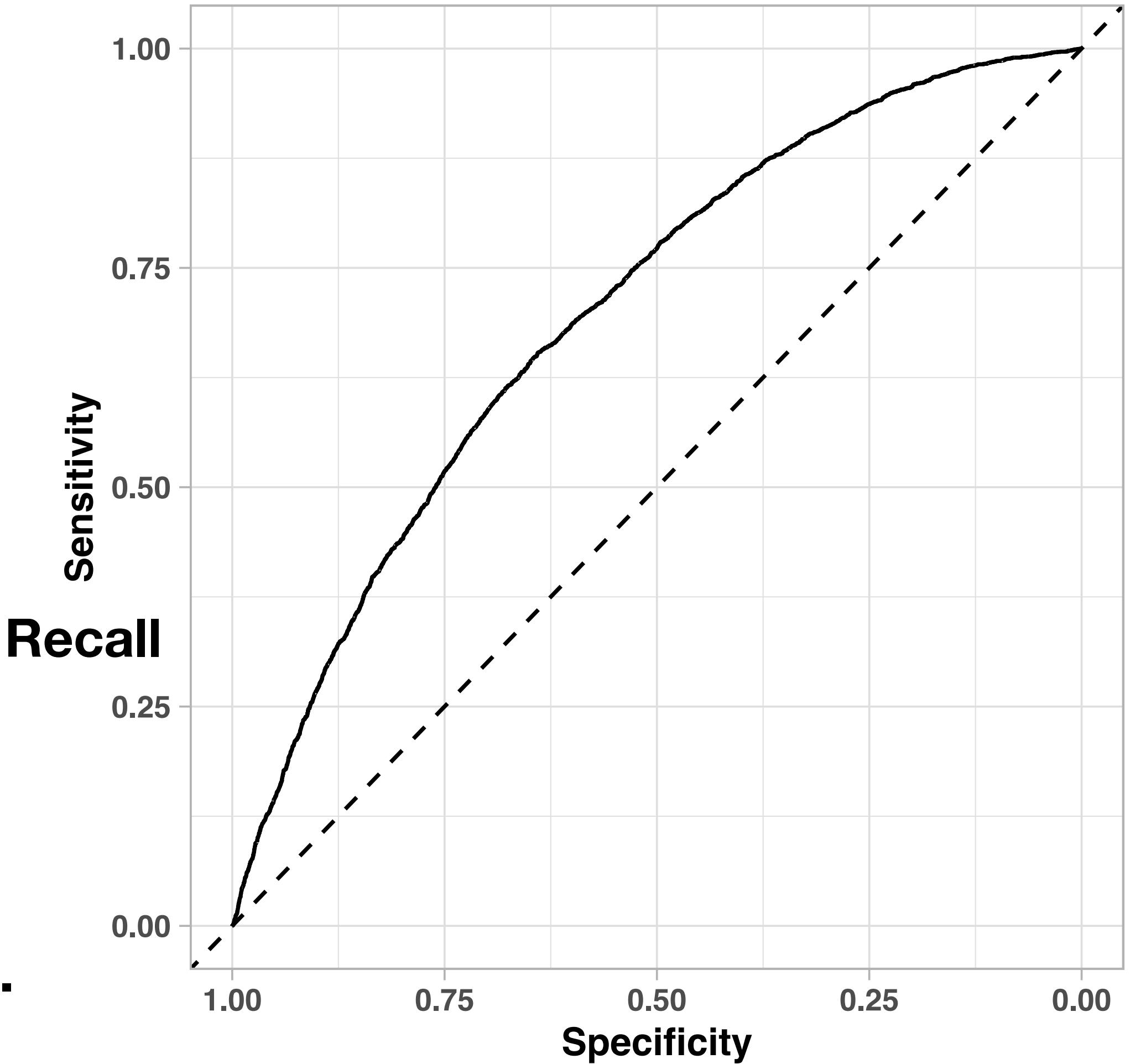


Me coming to terms with the possibility that I may have to consider an infinite number of thresholds.

There must be a better way.

```
# Generate ROC curve for training set  
roc_train <- roc(df_train$republican, df_train$predict)  
roc_train$auc  
  
Area under the curve: 0.6988  
  
# Plot ROC curve  
plot_3 <- ggroc(roc_train) +  
  theme_light() +  
  theme(text = element_text(size = 10, face = "bold")) +  
  geom_abline(slope=1, intercept=1, linetype="dashed") +  
  xlab("Specificity") +  
  ylab("Sensitivity")  
print(plot_3)
```

The ROC tests the performance of our model for all thresholds and summarizes that information in one graph.



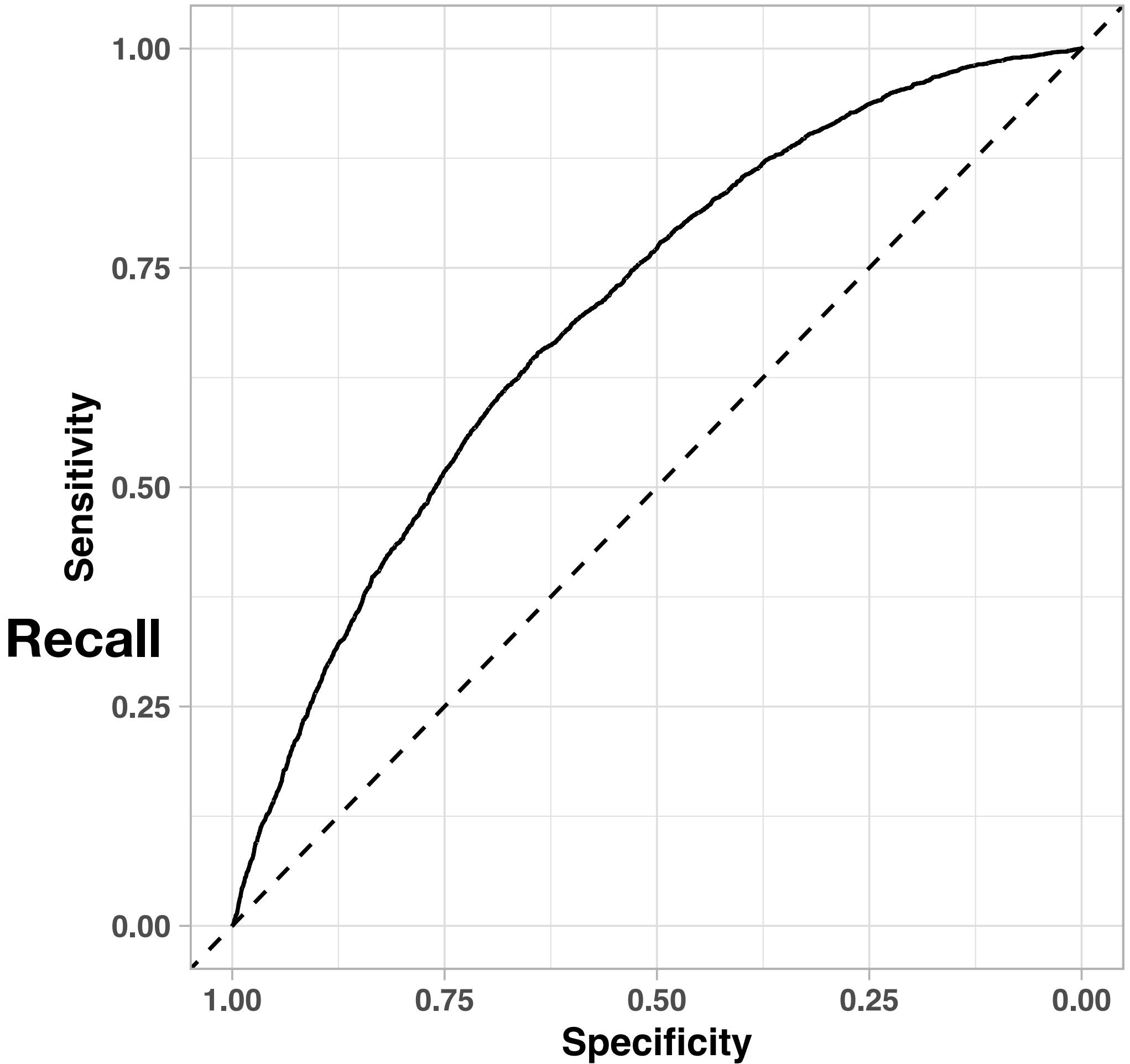
Sadly, not the same as precision.

There must be a better way.

```
# Generate ROC curve for training set  
roc_train <- roc(df_train$republican, df_train$predict)  
roc_train$auc  
  
Area under the curve: 0.6988  
  
# Plot ROC curve  
plot_3 <- ggroc(roc_train) +  
  theme_light() +  
  theme(text = element_text(size = 10, face = "bold")) +  
  geom_abline(slope=1, intercept=1, linetype="dashed") +  
  xlab("Specificity") +  
  ylab("Sensitivity")  
print(plot_3)
```

Area under the curve let's use numerically summarize the overall performance. It's the area under the ROC curve.

1 is perfect prediction. 0.5 is no better than chance.
Ours is about halfway between perfect and chance.



= Recall
Sadly, not the same as precision.

Let's evaluate the test set.

```
# Out-of-sample prediction  
df_test$predict <- predict(model_1, df_test)  
  
# Set potential thresholds  
df_test <- df_test %>% mutate(  
  threshold_1 = case_when(  
    predict > 0.4 ~ 1, TRUE ~ 0),  
  threshold_2 = case_when(  
    predict > 0.2 ~ 1, TRUE ~ 0  
)
```

```
# Evaluate thresholds  
table(df_test$republican, df_test$threshold_1)
```

	0	1
0	3225	646
1	947	540

```
table(df_test$republican, df_test$threshold_2)
```

	0	1
0	1497	2374
1	202	1285

Let's evaluate the test set.

```
# Out-of-sample prediction  
df_test$predict <- predict(model_1, df_test)  
  
# Set potential thresholds  
df_test <- df_test %>% mutate(  
  threshold_1 = case_when(  
    predict > 0.4 ~ 1, TRUE ~ 0),  
  threshold_2 = case_when(  
    predict > 0.2 ~ 1, TRUE ~ 0  
)  
  
# Evaluate thresholds  
table(df_test$republican, df_test$threshold_1)
```

	0	1
0	3225	646
1	947	540

```
table(df_test$republican, df_test$threshold_2)
```

	0	1
0	1497	2374
1	202	1285

For a threshold of 40%:

Precision

$$\begin{aligned} &= \text{True positives} / \text{All predicted positives} \\ &= 540 / (540 + 646) = 0.46 \end{aligned}$$

Recall (Sensitivity)

$$\begin{aligned} &= \text{True positives} / (\text{TP} + \text{FN}) \\ &= 540 / (540 + 947) = 0.36 \end{aligned}$$

For a threshold of 20%:

Precision

$$\begin{aligned} &= \text{True positives} / \text{All predicted positives} \\ &= 1285 / (1285 + 2374) = 0.35 \end{aligned}$$

Recall (Sensitivity)

$$\begin{aligned} &= \text{True positives} / (\text{TP} + \text{FN}) \\ &= 1285 / (1285 + 202) = 0.86 \end{aligned}$$

Let's evaluate the test set.

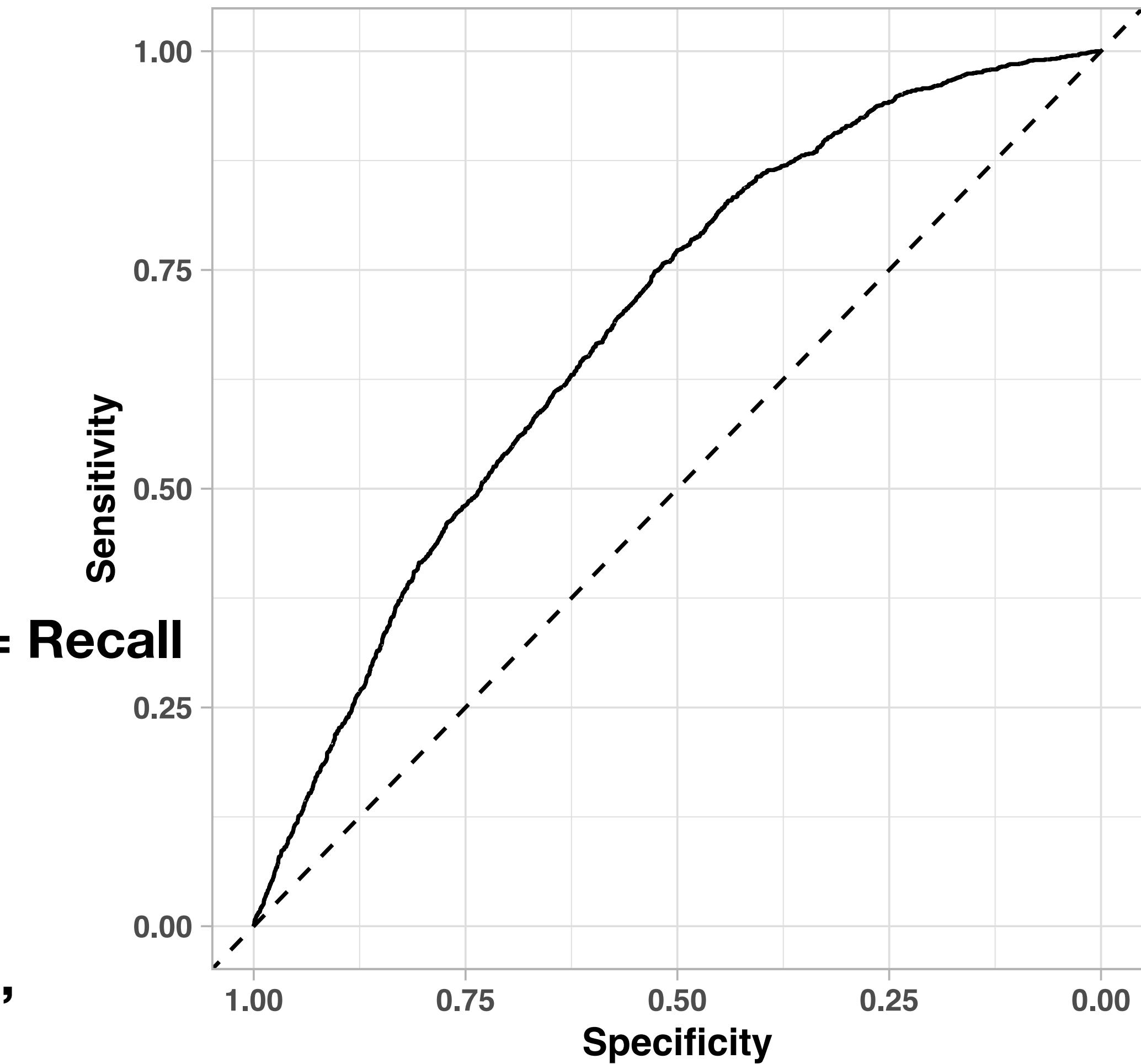
```
# Generate ROC curve for test set  
roc_test <- roc(df_test$republican, df_test$predict)  
roc_test$auc
```

Area under the curve: 0.6842

```
# Plot ROC curve  
plot_4 <- ggroc(roc_test) +  
  theme_light() +  
  theme(text = element_text(size = 10, face = "bold")) +  
  geom_abline(slope=1, intercept=1, linetype="dashed") +  
  xlab("Specificity") +  
  ylab("Sensitivity")  
print(plot_4)
```

The training set's AUC was 0.70.

So on the whole, we perform a little worse in the test set, which is to be expected, but not too much worse!



Predictions, Part 2.

We could enhance the model with LASSO.

It looks like we did last week; we just have to add `family = "binomial"` to the model.

But note that even for binary outcomes, the model itself generates a continuous “latent” prediction, so the principles are the all same.

We just evaluate binary vs. continuous models using different quantities (MSE vs. recall, etc.).

