# Lab 5

Ray Caraher

2025-04-21

# Setup

# Setting up our script

Before we get into any real coding, let's make sure that the preamble for our code looks good. Here is how I set it up:

```
## Load packages
library(haven)
library(tidyverse)
## Set options

options(scipen = 999)

## Clear environment

rm(list = ls())

## Set directories

base_directory <- '/Users/rcaraher/Library/CloudStorage/OneDrive-UniversityofMassachusetts/Academic/Teach
data_directory <- file.path(base_directory, 'Data')
results_directory <- file.path(base_directory, 'Results')
```

# Machine Learning in R

# Intro to Machine Learning

In this section fo the class, forget about **causality** (e.g, does the minimum wage cause unemployment?)

For now, focus on **prediction** (e.g., who is more likely to be a minimum wage worker?)

# What's the Goal?

- **Causal inference**
  - Estimate the effect of a policy/treatment (e.g. Y caused by D)
  - Answer "What if. . . ?" questions
- **Prediction**
  - Forecast Y given X
  - Answer "How well can I guess Y?"

# Identification vs. Generalization

| Aspect | Causal Inference | Prediction |
|---|---|---|
| Core focus | Identification of treatment effect | Minimizing prediction error |
| Required assumptions | Validity of research design | None; "data-driven" process |
| Evaluation metric | standard errors | "fit" statistics |
| "Cross-validation" | Placebo tests, sensitivity analysis | K-fold CV, hold-out test set |

# Combining Machine Learning and Causal Inference

Despite very different worlds, predictive tools from machine learning can be used in lots of creative ways for causal inference!

# From a Causal Question to a Prediction Problem

Recall rational for looking at the effect of minimum wage laws on teenagers:

- They are most likely to be directly affected by the policy (i.e., the sharpest "bite"), so we should focus on them to understand the effect of these policies

But the reality of **who** is likely to be a minimum wage worker is likely more complex than just age

- race, education, etc. are also likely to be important!
- Using a *single predictor* (age) may understate the likelihood someone is part of the group which may be most affected by the policy change

Now, instead of a causal problem, we have (for a minute) a **prediction problem**

# Problem Set Question

We want to predict **who** is most likely to be a minimum wage worker so we can study the effect of minimum wage policies *on that group*

We will use a basic machine learning exercise to do so

# Overview of Machine Learning

# 1. Problem Definition

- **Objective**: What are we trying to predict or learn?
- **Type of task**:
  - Supervised (regression, classification: we have a "Y" variable)
  - Unsupervised (clustering, dimensionality reduction: no "Y" variable; looking for patterns in the data)

# 2. Data Collection & Preparation

- **Inspect & clean**:
  - Handle missing values, outliers
  - Correct typos, standardize formats
- **Split** into training/test sets (e.g. 50/50, 80/20, etc.)

# 3. Feature (Variables) Engineering

- **Transform raw inputs** into features:
  - Scaling/normalization
  - Interaction terms, polynomial features
- **Dimensionality reduction** (if needed):
  - PCA, feature selection
- **Domain knowledge**:
  - Craft features informed by theory or context

# 4. Model Selection & Training

- **Choose candidate algorithms**
  - Linear models (logistic regression, LASSO), tree-based (RF, GBM), nueral nets
- **Hyperparameter tuning**
  - Cross validation to find optimal hyperparameters
- **Train models** on training set

# 5. Evaluation & Validation

- **Assess performance** on test sets
  - Regression: RMSE, R-squared
  - Classification: accuracy, AUC, precision/recall
- **Diagnostics**
  - Learning curves, bias–variance trade-off
- **Robustness checks**
  - Sensitivity to hyperparameters, data subsets

# 6. Deployment

- **Deploy model** to make predictions on “final” data

# Pipline for Our Problem

We want to:

1. **Get micro-data** from the CPS MORG data (contains individual-level demographic info)
2. **Pre-process** our "features" (i.e., variables) and data
3. **Split** our data into training/test sets
4. **Train** our model using a ML method on the training data (gradient boosting)
5. **Evaluate** our model using the test data (precision-recall curve)
6. **Deploy** our model on the full data
7. **Calculate** employment and wage rates for likely minimum wage workers
8. **Estimate** the effect of CA's MW increase on the employment/wage rates of those workers

# MORG data

Let's read in the MORG data and take a look at it

```
morg <- read_dta(file.path(data_directory, "CPSmorg_1979_1990_small.dta"))

glimpse(morg)
```

```
## Rows: 4,126,876
## Columns: 16
## $ month       <dbl+lbl>  4,  4, 10,  9,  8,  8,  8,  2, 12,  4,  8,  6,  5,
## $ hhid        <chr> "301093021702", "808342401006", "808012816108", "7320932
## $ state       <dbl+lbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
## $ age         <dbl> 72, 57, 61, 33, 34, 32, 31, 36, 37, 39, 28, 64, 30, 27,
## $ race        <dbl+lbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
## $ sex         <dbl+lbl> 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0
## $ esr         <dbl+lbl>  4,  4,  4,  1,  3,  2,  1, NA,  1,  1, NA,  4,  1,
## $ earnhre     <dbl> NA, NA, NA, NA, NA, 420, 375, 395, 1111, NA, NA, NA, NA,
## $ year        <dbl> 1981, 1980, 1981, 1987, 1980, 1980, 1981, 1989, 1985, 19
## $ statenum    <dbl+lbl> 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23,
## $ cpi         <dbl> 38.79457, 35.28579, 40.29428, 49.91511, 36.38936, 36.389
## $ hispanic    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ dmarried    <dbl> 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1,
## $ ruralstatus <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 1, 2,
## $ veteran     <dbl> 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
## $ educcat     <dbl> 1, 3, 1, 4, 2, 2, 2, 3, 4, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2,
```

# Feature engineering

Let's now process our data to get it ready for our ML technique

```
## Define minimum wage worker

morg <- morg |>
  mutate(mw = case_when(earnhre > 100 & earnhre < 335 ~ 1,
                        is.na(earnhre) ~ NA_real_,
                        TRUE ~ 0))

count(morg, mw)
```

```
## # A tibble: 3 x 2
##      mw       n
##   <dbl>   <int>
## 1     0 1190027
## 2     1   90077
## 3    NA 2846772
```

```r
### Convert to factors

morg <- morg |>
  mutate(mw.f = case_when(mw == 1 ~ "y",
                          mw == 0 ~ "n",
                          is.na(mw) ~ NA_character_),
         mw.f = as.factor(mw.f),
         mw.f = relevel(mw.f, ref = "y"))

morg <- morg |>
  mutate(ruralstatus.f = factor(ruralstatus),
         sex.f = factor(sex),
         race.f = factor(race),
         educcat.f = factor(educcat),
         hispanic.f = factor(hispanic),
         married.f = factor(dmarried),
         vet.f = factor(veteran))

morg <- morg |>
  arrange(hhid, year, month) |>
  mutate(rowid = 1:n())

## Create data subset

morg_pre <-  morg |>
  filter(year >= 1985 & year <= 1987)
```

# Notes about feature engineering

- You will need to be aware of how the ML package you are using prefers to have the data
- Generally, you will want to convert categorical variables to factors and omit missing rows
- The best implementation of ML pipelines in R are from the `tidymodels` set of packages

# Training Data

Let's now sample our data and select the rows that we will use to train our model

```
## Select training data

morg_pp <- morg_pre |>
  select(rowid, mw, educcat.f, ruralstatus.f, sex.f, race.f, hispanic.f, marrie
  na.omit()

set.seed(81781567)
morg_train <- morg_pp |>
  slice_sample(prop = 0.20)

morg_test <- morg_pp |>
  filter(!(rowid %in% morg_train$rowid))
```

# Training the Model

Now that we have processed the data and selected our training sample, we can now estimate the parameters.

We will use **gradient boosting** here!

```
#install.packages("gbm)
library(gbm)
```

```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning t
```

```
## Run GBM

gbm1 <- gbm(mw ~ educcat.f + ruralstatus.f + sex.f + hispanic.f + married.f + r
            data = morg_train)
```

```
## Distribution not specified, assuming bernoulli ...
```

# Gradient Boosting

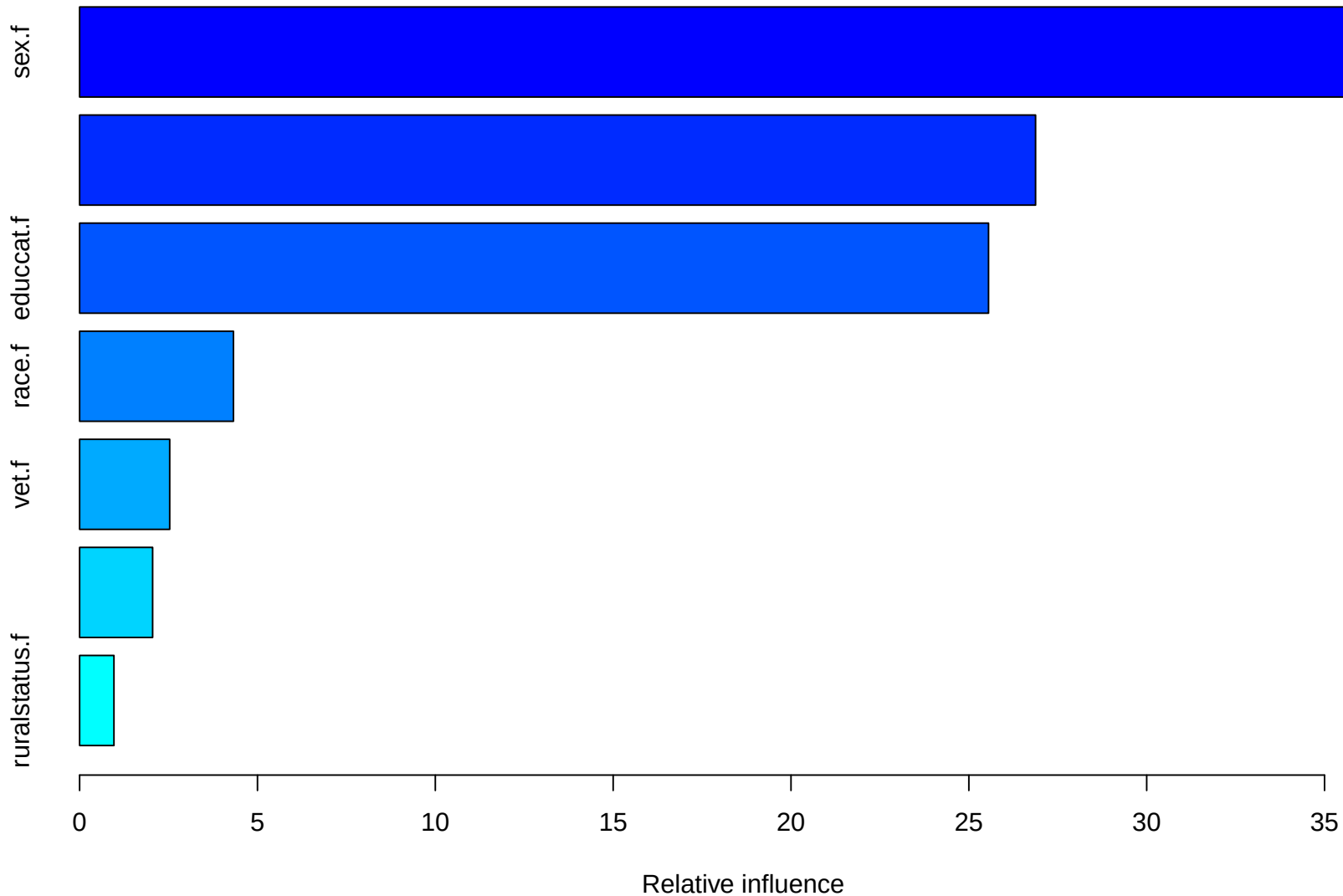
- **Core idea:**
  Build a strong predictor by *sequentially* adding many weak learners (usually small decision trees), each one correcting the mistakes of the ensemble so far.

- **How it works:**

  1. **Initialize** with a very simple model (e.g., predict the average outcome for everyone).
  2. **Measure errors:** For each observation, see how far off the current model's prediction is from the actual value.
  3. **Learn from mistakes:** Train a new, shallow decision tree to predict those errors.
  4. **Update the model:** Add the new tree to your ensemble, but scale its contribution by a small factor (the learning rate) so you don't overcorrect.
  5. **Repeat:** Use the updated ensemble to identify remaining errors, fit another tree to those, and add it in. After many rounds, the combined trees form the final predictor.

# Gradient Boosting: Key Variables

```
summary(gbm1)
```




```
##                          var    rel.inf
## sex.f                  sex.f 37.6952435
## married.f          married.f 26.8758977
```

# Validating the Data

Now that we have trained the model, we want to look at how well it did.

We will first need to use `predict()` to generate the predicted probabilities on the test dataset.

```
morg_test <- morg_test |>
  mutate(pred = predict(gbm1, newdata = morg_test, type = "response"))
```

```
## Using 100 trees...
```

```
fivenum(morg_test$pred)
```

```
## [1] 0.001346357 0.009676978 0.019467815 0.035153008 0.111161584
```

# Evaluating our model

Now that we have our predictions, we want to see how well we did

There are **many** different ways and metrics we can choose to do this with

Which one is *right* depends on the context of your problem (i.e., how costly are certain types of missclassifications)

# Confusion Matrix

| | **Predicted Positive** | **Predicted Negative** |
|---|---|---|
| **Actual Positive** | TP | FN |
| **Actual Negative** | FP | TN |

# Precision and Recall

**Precision**

Of all instances predicted *positive*, the share that are truly positive.

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

**Recall (Sensitivity)**

Of all truly *positive* instances, the share correctly identified.

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

- **High Precision** $\Rightarrow$ Few false positives (when the model says "yes," it's usually right).
- **High Recall** $\Rightarrow$ Few false negatives (the model finds most of the true positives).

# Threshold Selection

To turn a probability score into a class label, we choose a **threshold** $\tau$:

$$\hat{y} = \begin{cases} 1, & p \geq \tau, \\ 0, & p < \tau. \end{cases}$$

- **Raising the threshold**
  - Fewer predicted positives $\Rightarrow$ **Higher precision** ($\downarrow \mathrm{FP}$)
  - More missed positives $\Rightarrow$ **Lower recall** ($\uparrow \mathrm{FN}$)
- **Lowering the threshold**
  - More predicted positives $\Rightarrow$ **Higher recall** ($\downarrow \mathrm{FN}$)
  - More false alarms $\Rightarrow$ **Lower precision** ($\uparrow \mathrm{FP}$)
- **Visualizing the trade-off:**
  - **Precision–Recall curve:** plot precision vs. recall as $\tau$ varies.

# Calculating Metrics at the Default Threshold

Let's compute recall and precision of our model at the default threshold (0.50)

```
morg_test <- morg_test |>
  mutate(class = case_when(pred > 0.5 ~ 1,
                           TRUE ~ 0))

confusion_mat <- count(morg_test, class, mw)

tp <- morg_test |>
  filter(class == 1 & mw == 1) |>
  summarise(tp = n()) %>%
  pull(tp)

fn <- morg_test |>
  filter(class == 0 & mw == 1) |>
  summarise(fn = n()) %>%
  pull(fn)

fp <- morg_test |>
  filter(class == 1 & mw == 0) |>
  summarise(fp = n()) %>%
  pull(fp)
```

# Calculating Metrics at the Default Threshold

```
recall <- tp / (tp + fn)

percision <- tp / (tp + fp)

recall
```

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

```
percision
```

```
## [1] NaN
```

# Calculating Metrics at Many Thresholds

This will help us choose the value of recall and precision that provides the best balance (or other criteria)

Let's compute these values for many different thresholds and plot the **precision-recall curve**

# Calculating Metrics at Many Thresholds

```
pr_curve_tab <- tibble()

for (i in seq(0.0001, 0.9999, 0.0001)) {
  thresh <- i
  morg_test <- morg_test |>
    mutate(class = case_when(pred > thresh ~ 1,
                             TRUE ~ 0))
  tp <- morg_test |>
    filter(class == 1 & mw == 1) |>
    summarise(tp = n()) %>%
    pull(tp)
  fn <- morg_test |>
    filter(class == 0 & mw == 1) |>
    summarise(fn = n()) %>%
    pull(fn)
  fp <- morg_test |>
    filter(class == 1 & mw == 0) |>
    summarise(fp = n()) %>%
    pull(fp)
  recall <- tp / (tp + fn)
  perc <- tp / (tp + fp)
  tab <- tibble(thresh, recall, perc)
  pr_curve_tab <- bind_rows(pr_curve_tab, tab)
}
```

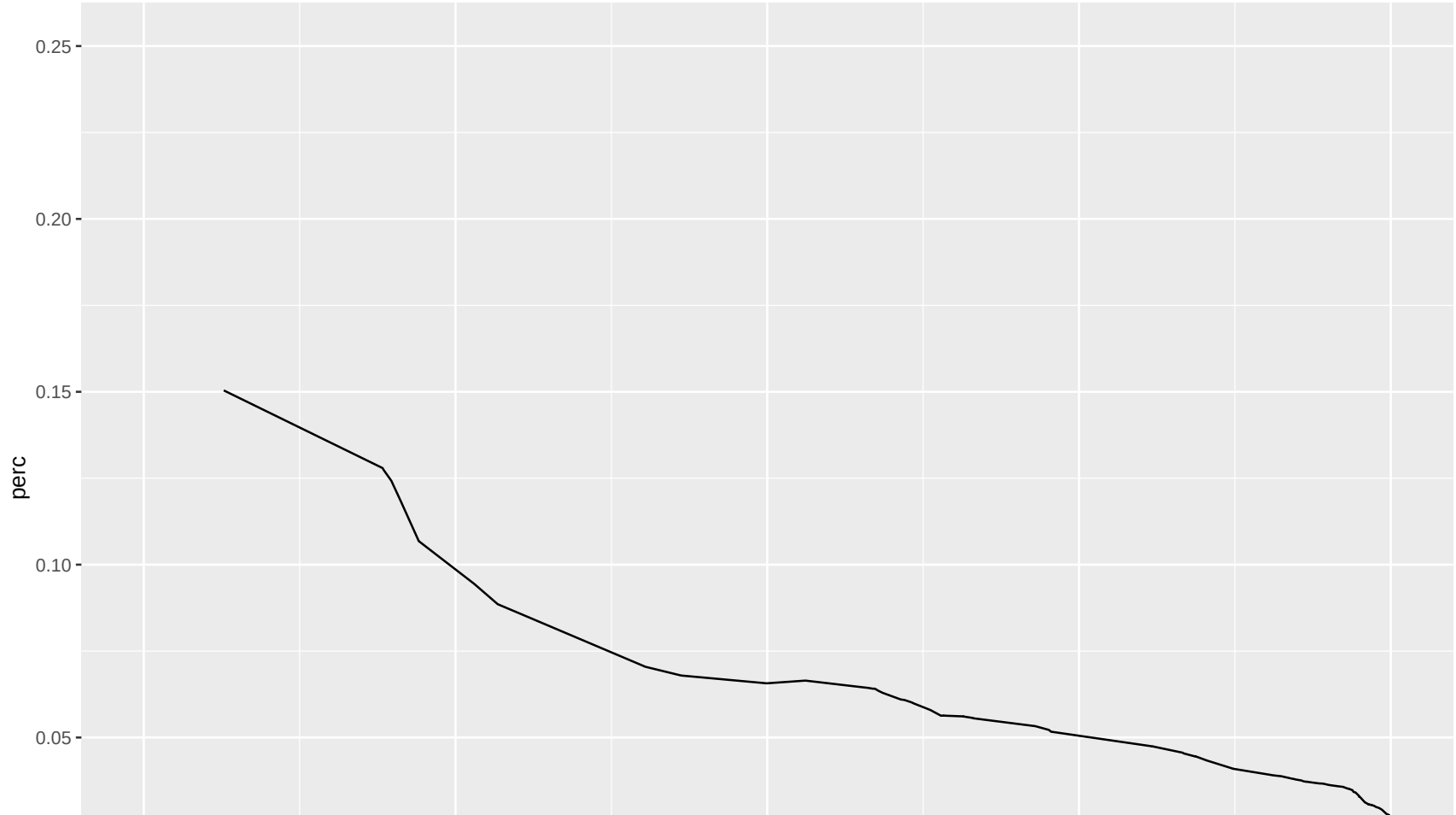
# Plotting the PR Curve

```
## Print curve

p1 <- ggplot(pr_curve_tab) +
  geom_path(aes(x = recall, y = perc)) +
  coord_cartesian(ylim = c(0, 0.25))

print(p1)
```

```
## Warning: Removed 8888 rows containing missing values or values outside the scale range
## (`geom_path()`).
```



# The High-Recall Group

Let's say that we want to optimize the recall rate to be about 75%

Let's find the point on our curve where this holds and the associated threshold value

```
pr_curve_tab %>%
  filter(recall > 0.74 & recall < 0.76)
```

```
## # A tibble: 0 x 3
## # i 3 variables: thresh <dbl>, recall <dbl>, perc <dbl>
```

```
final_threshold <- 0.025
```

# Predicting the outcome for the full data

Now that we have trained our model and chosen the appropriate threshold, we can finally generate the predicted values for our full dataset

```
morg <- morg |>
  mutate(pred_class = predict(gbm1, newdata = morg, type = "response"))
```

```
## Using 100 trees...
```

```
morg <- morg |>
  mutate(pred_mw = case_when(pred_class > final_threshold ~ 1,
                             is.na(pred_class) ~ NA_real_,
                             TRUE ~ 0))
morg |>
  group_by(pred_mw) |>
  summarise(n = n(),
            mean_wage = mean(earnhre, na.rm = T))
```

```
## # A tibble: 2 x 3
##   pred_mw       n mean_wage
##     <dbl>   <int>     <dbl>
## 1       0 2740097      772.
## 2       1 1386779      519.
```