Review Session 10

Machine Learning

Ben Berger

April 14, 2023

Today's Agenda

- Review process of machine learning.
- Extended R exercise.

Step 1: Clean and transform variables in dataset

What data we include in our dataset determines what data is available for an algorithm to use. This requires us to make choices.

Should you log, normalize, or transform your data? Do you include all variables? Do you drop those that are highly correlated? Do you recenter or rescale variables? How would you think through these choices?

Step 2: Pick your functions and regularizers

There are many ML methods to choose from. How do we pick?

Some considerations:

1. Type of outcome: continuous vs. categorical
2. Linearity of relationship between predictors and outcome
3. Computational time/costs
4. Need for interpretability
5. Not much theoretical guidance here; instead, it is often treated as an empirical question $\rightarrow$ use what works best.

On next problem set, we focus on using simple linear regression, kitchen sink regression and LASSO.

Step 3: Split your data into training and testing sets

How big should your training set be? How big should your test set be? Bigger is better, but how do you know when your data is "big enough?"
Some Considerations:

1. Answer could depend on many things, including the complexity of the underlying data and relationship between predictors and outcome, how sparse your data is (e.g. if few observations have similar sets of characteristics or your outcome is rare), size of your irreducible error, etc.
2. In general, you don't want to see large changes in your chosen model due to slight changes in size of training set. This would be a sign of overfitting and that your training set is not big enough.

Step 4: Use CV to find optimal tuning parameter(s)

How many folds? Which tuning parameter? How to decide?

Considerations:

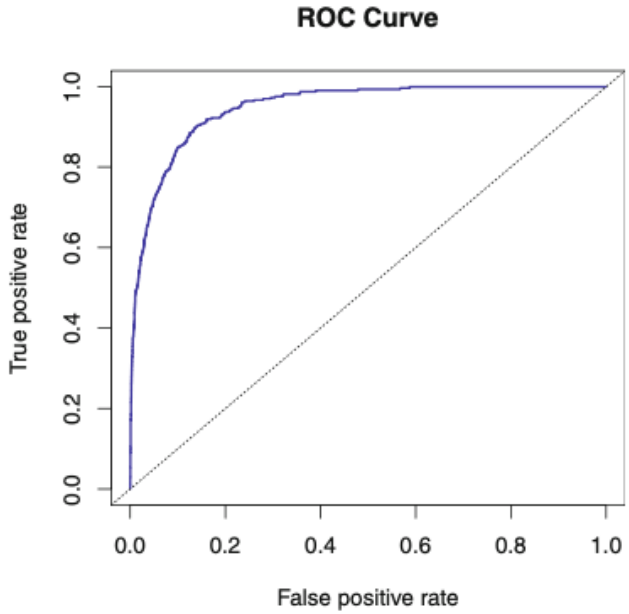
1. For folds: convention is 5 or 10, but why? Extreme uses as many folds as there are observations in the training set (this is known as Leave-One-Out Cross-Validation, LOOCV). While adding more folds initially reduces both the bias and variance of our chosen model, variance eventually increases because the folds become correlated with each other $\rightarrow$it's the inverted U-shape due to the bias variance tradeoff again!
2. For tuning parameters: choose the one that minimizes MSE.

For the next Pset, we use 10 folds (default of `cv.glmnet`) and select the `lambda.min` from the output of `cv.glmnet`.

Step 5: For classification, select cutoff value

How do we chose the best threshold value?

- Select value closest to upper-left-hand corner of ROC curve, unless you have substantive reasons for selecting another point.



For the next Pset, we use the the 70% quantile and above (see, Consideration 3, pages 10-11). The optional question A5 works through making ROC curve.

Step 6: Compute performance metric

How do we assess the performance of regression and classification models?

1. Regression: MSE on the test set
2. Classification: precision, recall, confusion matrices, and others

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

Figure 1: Confusion Matrix

Step 7: Pick the best one, then make predictions

What if there is no clear winner?

Considerations:

- Use substantive knowledge
- Models will also differ complexity, interpretability, and other attributes

Exercise: Programming ML in R – Introduction

As an introduction to doing machine learning in R, we are going to use data on Boston house prices from the 1970 census. Each observation is a census tract and the dependent variable `medv` is the median value of owner-occupied homes within the tract in thousands of dollars.

We want to predict `medv` using the other variables like `crim` (crime rate), `rm` (average number of rooms per home), and `ptratio` (student-teacher ratio.)

If you're following along, make sure to first install the packages `rsample`, `glmnet`, `mlbench`, and `pROC`. Then load the data using `data(BostonHousing)`.

```
install.packages(c("rsample", "glmnet", "mlbench", "pROC"))
```

Exercise: Programming ML in R – Setup

```
library(tidyverse)
library(broom)
library(rsample) # data splitting
library(glmnet) # LASSO estimation
library(mlbench) # BostonHousing dataset
library(pROC) # ROC curves

# Load data
data(BostonHousing)
BostonHousing %>% select(medv, 1:11) %>% head
```

```
  medv    crim zn indus chas   nox
1 24.0 0.00632 18  2.31    0 0.538
2 21.6 0.02731  0  7.07    0 0.469
3 34.7 0.02729  0  7.07    0 0.469
4 33.4 0.03237  0  2.18    0 0.458
5 36.2 0.06905  0  2.18    0 0.458
6 28.7 0.02985  0  2.18    0 0.458
     rm  age    dis rad tax ptratio
1 6.575 65.2 4.0900   1 296    15.3
2 6.421 78.9 4.9671   2 242    17.8
3 7.185 61.1 4.9671   2 242    17.8
4 6.998 45.8 6.0622   3 222    18.7
5 7.147 54.2 6.0622   3 222    18.7
```

Exercise: Programming ML in R – Sample Splitting

Now we need to split the sample. Do this using `initial_split()`, `training()`, and `testing()` from the package `rsample`.

```
# Split Sample
set.seed(1)
split <- initial_split(BostonHousing, prop = 0.7)
boston_train <- training(split)
boston_test <- testing(split)
```

Exercise: Programming ML in R – OLS Estimation

Just because we are doing machine learning doesn't mean we estimate OLS any different.

Well actually we don't care about getting robust std. errors for coefficients now, so we can just use `lm` to estimate linear regression via OLS.

```
# Define model formula
medv_formula <- medv ~ crim + zn + indus + chas + nox +
  rm + age + dis + rad + tax + ptratio + b + lstat

# OLS -- Regress median housing value on all other variables
fit_ols <- lm(medv_formula, boston_train)
```

Exercise: Programming ML in R – LASSO Estimation

Unfortunately we can't directly use the formula notation for `cv.glmnet`. Instead we need to provide an independent variable matrix and dependent variable vector.

Also, LASSO will estimate a model with two intercepts if we don't remove the first column in this dataset first. Annoying!

```
# Extract independent variables in training set + remove column 1
X_train <- model.matrix(medv_formula, boston_train)
X_train <- X_train[,-1]

# While we're here... extract independent variables for test set
X_test <- model.matrix(medv_formula, boston_test)
X_test <- X_test[,-1]
```

Exercise: Programming ML in R – LASSO Estimation

To estimate a regression model using LASSO with a cross-validated tuning parameter use `cv.glmnet` from the package `glmnet`. This function automatically splits the training sample to evaluate many different penalites λ.

Syntax: `fit_lasso <- cv.glmnet(X, y)` where `X` is a matrix and `y` is a vector.

```
# Estimate LASSO (choosing lambda by cross-validation)
fit_lasso <- cv.glmnet(X_train, boston_train$medv)
```

Exercise: Programming ML in R – LASSO Output

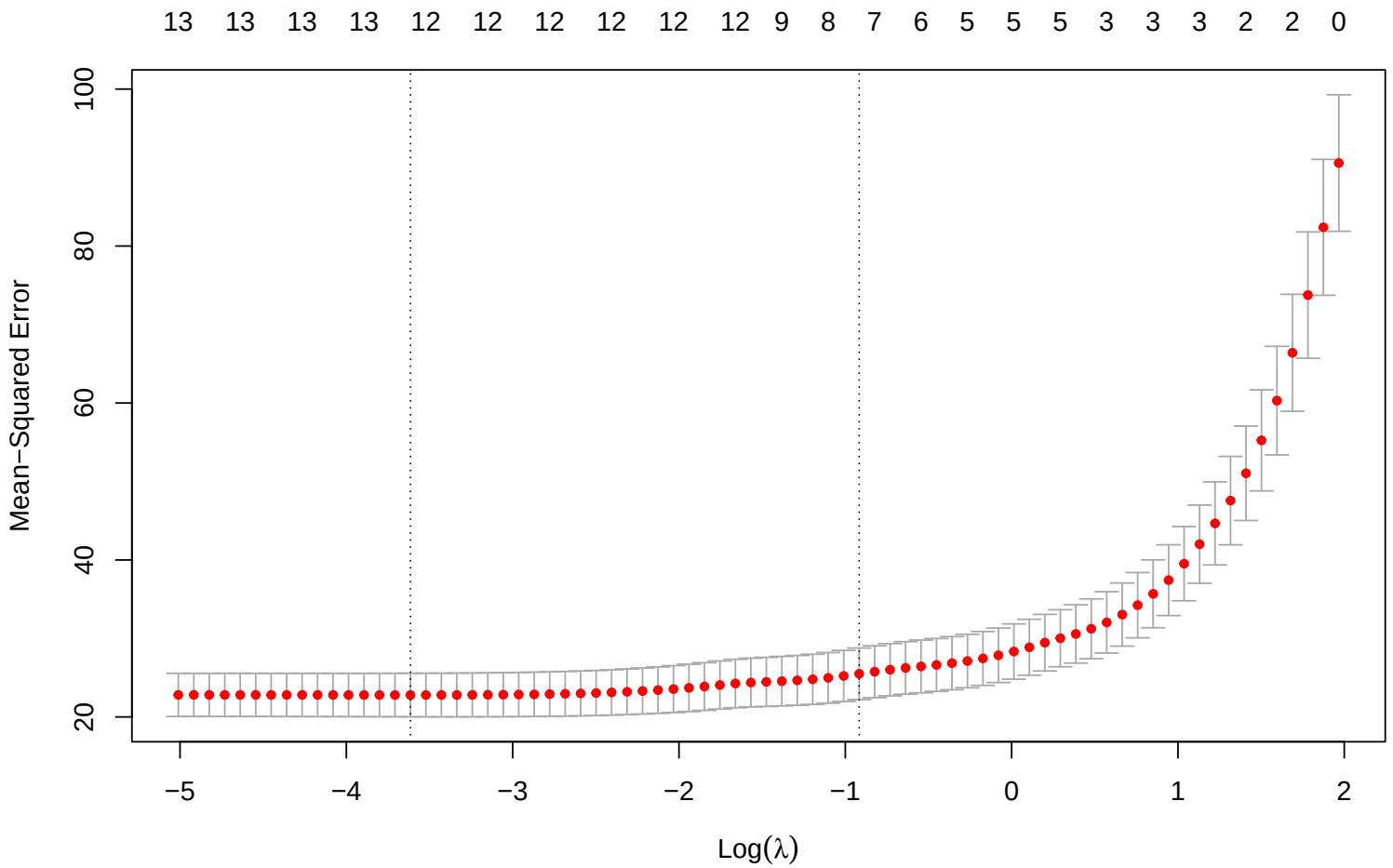
```
# Save LASSO penalties and performance
cv_lasso_values <- fit_lasso %>%
  tidy() %>%
  select(lambda, cv_mse = estimate)

head(cv_lasso_values)
```

```
# A tibble: 6 x 2
  lambda cv_mse
   <dbl>  <dbl>
1   7.15   90.6
2   6.52   82.4
3   5.94   73.7
4   5.41   66.4
5   4.93   60.3
6   4.49   55.2
```

Exercise: Programming ML in R – LASSO Output

```
plot(fit_lasso)
```



Exercise: Programming ML in R – Variable Selection

```
tibble(term = names(fit_ols$coefficients),
       ols_coef = coef(fit_ols),
       lasso_coef = coef(fit_lasso, s = "lambda.min")[, 1])
```

```
# A tibble: 14 x 3
   term        ols_coef lasso_coef
   <chr>          <dbl>      <dbl>
 1 (Intercept)  28.2       26.3
 2 crim         -0.0706    -0.0614
 3 zn            0.0360     0.0315
 4 indus         0.0301     0
 5 chas1         3.42       3.45
 6 nox         -14.6      -13.2
 7 rm            4.86       4.90
 8 age          -0.0145    -0.0129
 9 dis          -1.41      -1.33
10 rad           0.304      0.254
11 tax          -0.0137    -0.0113
12 ptratio      -0.942     -0.920
13 b             0.0107     0.0104
14 lstat        -0.480     -0.478
```

Exercise: Programming ML in R – Calculating Predictions

```
# Get predictions ------------- #
# Training set
boston_train <- mutate(boston_train,
                       pred_ols = predict(fit_ols, boston_train),
                       pred_lasso = predict(fit_lasso, X_train,
                                            s = "lambda.min"))

# Test set
boston_test <- mutate(boston_test,
                      pred_ols = predict(fit_ols, boston_test),
                      pred_lasso = predict(fit_lasso, X_test,
                                           s = "lambda.min"))
```

Exercise: Programming ML in R – Mean Squared Error

Now for the training and test set, calculate the mean squared error for both OLS and LASSO.

```
# Training Set
boston_train %>%
  summarize(mse_ols = mean((medv - pred_ols)^2),
            mse_lasso = mean((medv - pred_lasso)^2))
```

```
  mse_ols mse_lasso
1 20.2592  20.30264
```

```
# Test Set
boston_test %>%
  summarize(mse_ols = mean((medv - pred_ols)^2),
            mse_lasso = mean((medv - pred_lasso)^2))
```

```
   mse_ols mse_lasso
1 27.31196  27.32383
```

Classification

We use classification whenever we have a categorical dependent variable.

Models & Regularizers:

- Logistic regression with LASSO (or other) regularizer
- Classification trees (or forests) with "pruning" determined by tuning parameter α

Cutoff Values: Logistic regression produces predictive values $\rightarrow$How do we convert predictive values to categorical outcomes?

1. ROC curves
2. Context specific considerations (e.g. cost of false positives vs. false negatives, etc.)

Binary Outcome Models

Let y_i be a binary outcome variable, X_i a vector of explanatory variables, and $\beta = (\beta_0, ..., \beta_P)$ the vector of coefficients. We can use a generalized linear model like logit or probit to estimate $\Pr(y_i = 1|X_i; \beta)$.

$$\Pr(y_i = 1|X_i; \beta) = f(\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + ... + \beta_P X_{Pi})$$

A link function f transforms the linear part of the model to probabilities that are constrained between 0 and 1.

- If $f(a) = \Phi(a)$ (a normal CDF) then this is a probit model.
- If $f(a) = \frac{1}{1+exp(-a)}$ then this is a logit model (logistic regression).

Logistic Regression in ML

In the machine learning context, we normally estimate the logistic regression model.

To estimate a logit model, we use maximum likelihood, or equivalently, we minimize the negative of the log likelihood:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i \log \Pr(y_i = 1|X_i;\beta) + (1-y_i)\log \Pr(y_i = 0|X_i;\beta)\Big)$$

Suppose X_i contains many variables, and thus minimizing the above equation runs the risk of overfitting. How can we modify Equation 1 to control overfitting? Add a regularizer like LASSO!

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i \log \Pr(y_i = 1|X_i;\beta) + (1-y_i)\log \Pr(y_i = 0|X_i;\beta)\Big) + \lambda\sum_{j=0}^{P}|\beta_j|$$

Classification

Evaluative Criteria:

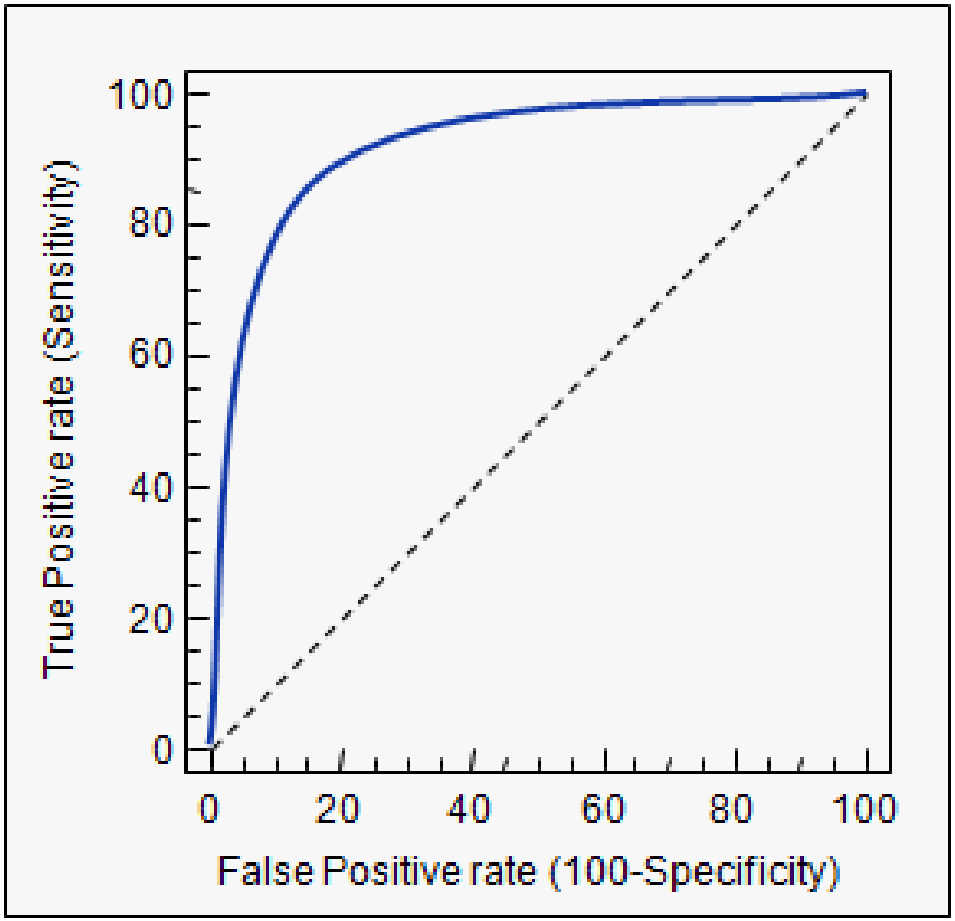
$$\text{Precision} = \frac{\text{True Positives}}{\text{All Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Question:

- We have two algorithms that predict heart attack risk which we use to target a physician-based intervention. Algorithm A has higher precision but lower recall. Algorithm B has lower precision but higher recall.
  - If we want to maximize the fraction of people who receive the intervention that are truly at high risk, which algorithm should we use? *Algorithm A*
  - What if we want to maximize the fraction of people at high risk that receive the intervention? *Algorithm B*

ROC Curve



- Points close to the upper left corner of the ROC curve have a high true positive rate and a low false positive rate, so choosing a cutoff point there for classification is appealing.
- However, based on how the predictions will be used you may want to be more/less careful about cutoff location (e.g. bail decisions — do we even want to use ML for this?).

Exercise: Programming ML in R – Binary Outcomes

Suppose Boston Mayor Michelle Wu proposes paying for a new policy with a property tax increase on houses in census tracts that had median home value of $35,000 or greater in 1970. Random!

We want to predict whether property owners in census tracts will be exposed to the new tax.

Exercise: Programming ML in R – Binary Outcomes

Create a new variable called `newtax` in the training and test datasets that is equal to 1 if median housing value $\geq \$35,000$ and 0 otherwise. Recall that `medv` is measured in thousands of dollars.

```
# Add binary indicator for exposure to new tax
boston_train <- mutate(boston_train,
                       newtax = ifelse(medv >= 35, 1, 0))
boston_test <- mutate(boston_test,
                      newtax = ifelse(medv >= 35, 1, 0))
```

Exercise: Programming ML in R – Classification

Now we want to fit classification models with logit and LASSO-logit.

To estimate logit, run a command like `glm(formula, data, family = "binomial")`. The option `family = "binomial"` tells `glm` to estimate a logit model. Keep in mind that our formula has a different dependent variable now!

To estimate LASSO-logit, use the function `cv.glmnet()` like before, but now additionally specify the option `family = "binomial"`.

Exercise: Programming ML in R – Classification

```
# Define formula
newtax_formula <- newtax ~ crim + zn + indus + chas + nox +
  rm + age + dis + rad + tax + ptratio + b + lstat

# Logit
fit_logit <- glm(newtax_formula, boston_train,
                 family = "binomial")

# LASSO-Logit
fit_lassologit <- cv.glmnet(X_train, boston_train$newtax,
                            family = "binomial")
```

Exercise: Programming ML in R – Predicted Probabilities

Now calculate predicted probabilities for the training and test sets and add them to the original datasets. Call them `prob_logit` and `prob_lassologit`.

Logit:

```
predict(model, training_data, type = "response")
```

LASSO logit:

```
predict(model, training_matrix, type = "response", s = "lambda.min")[, 1]
```

Exercise: Programming ML in R – Predicted Probabilities

```
# Calculate predicted probabilities ------------ #
# Training set
boston_train <- mutate(boston_train,
                       prob_logit = predict(fit_logit, boston_train,
                                            type = "response"),
                       prob_lassologit = predict(fit_lassologit, X_train,
                                                 type = "response",
                                                 s = "lambda.min")[, 1])
boston_test <- mutate(boston_test,
                      prob_logit = predict(fit_logit, boston_test,
                                           type = "response"),
                      prob_lassologit = predict(fit_lassologit, X_test,
                                                type = "response",
                                                s = "lambda.min")[, 1])
```

Exercise: Programming ML in R – Setting the Cutoff

Now we use the predicted probabilities to assign predictions to the training and test sets. Create new variables `pred_logit` and `pred_lassologit` that are equal to 1 if their probability is greater than the 75th percentile of probabilities and 0 otherwise. You can calculate the 75th percentile of `x` using `quantile(x, 0.5)`.

```
# Set cutoffs  ------------ #
boston_train <- boston_train %>%
  mutate(pred_logit = ifelse(prob_logit > quantile(prob_logit, 0.75),
                             1, 0),
         pred_lassologit = ifelse(prob_lassologit > quantile(prob_lassologit, 0.75),
                                  1, 0))

boston_test <- boston_test %>%
  mutate(pred_logit = ifelse(prob_logit > quantile(prob_logit, 0.75),
                             1, 0),
         pred_lassologit = ifelse(prob_lassologit > quantile(prob_lassologit, 0.75),
                                  1, 0))
```

Exercise: Programming ML in R – Evaluating Performance

Calculate the precision and recall in the training and test samples.

$$\text{Precision} = \frac{\text{Tax-Liable and Predicted Liable}}{\text{Predicted Liable}}$$

$$\text{Recall} = \frac{\text{Tax-Liable and Predicted Liable}}{\text{Tax-Liable}}$$

Exercise: Programming ML in R – Evaluating Performance

```
# Training set: precision
summarize(boston_train,
          precision_logit = sum(newtax * pred_logit) / sum(pred_logit),
          precision_lassologit = sum(newtax * pred_lassologit) / sum(pred_lassologit))
```

```
  precision_logit
1       0.4269663
  precision_lassologit
1            0.4157303
```

```
# Training set: recall
summarize(boston_train,
          recall_logit = sum(pred_logit * newtax) / sum(newtax),
          recall_lassologit = sum(pred_lassologit * newtax) / sum(newtax))
```

```
  recall_logit recall_lassologit
1            1         0.9736842
```

Exercise: Programming ML in R – Evaluating Performance

```
# Test set: precision
summarize(boston_test,
          precision_logit = sum(newtax * pred_logit) / sum(pred_logit),
          precision_lassologit = sum(newtax * pred_lassologit) / sum(pred_lassologit))
```

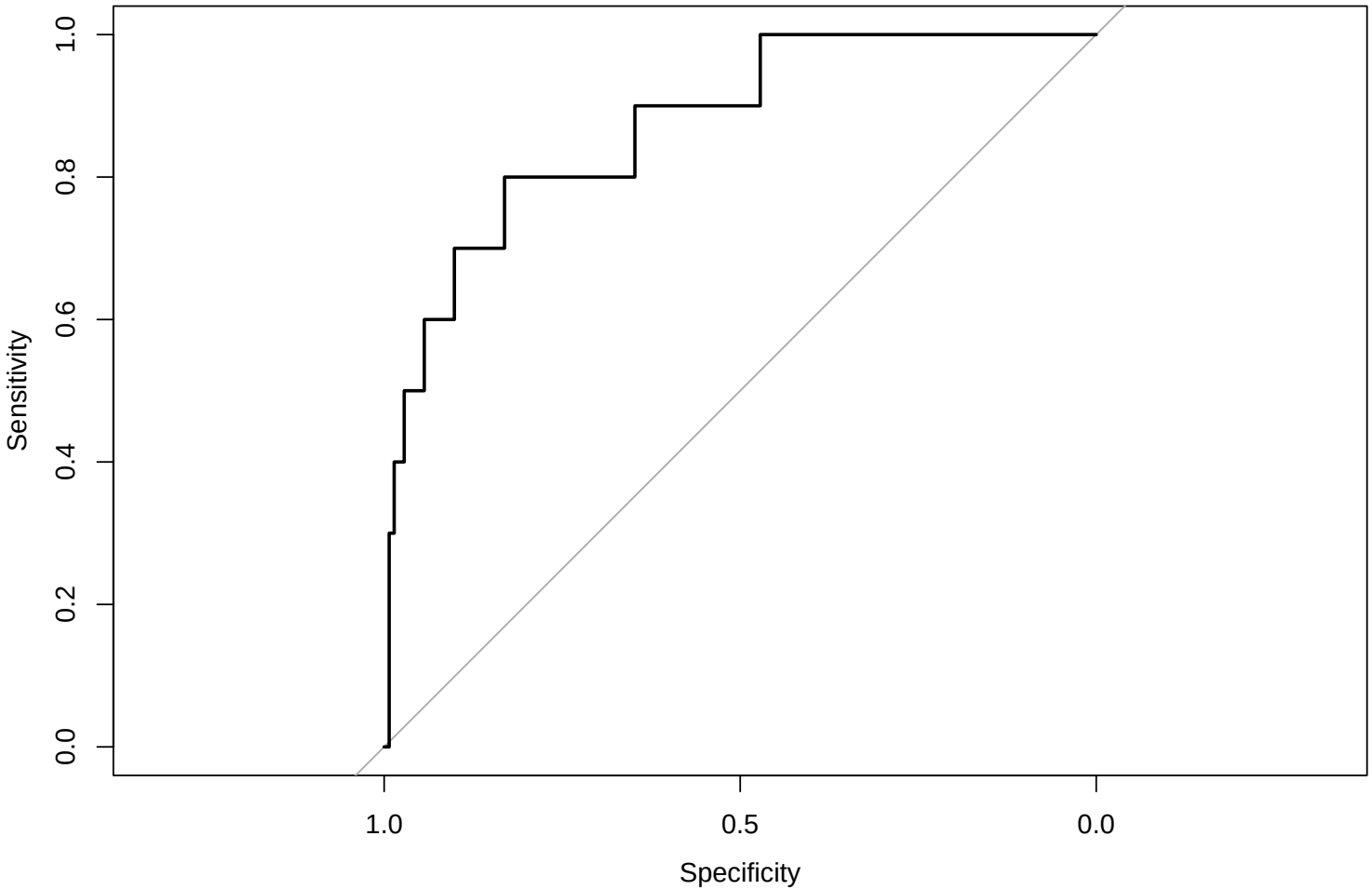
```
  precision_logit
1       0.2368421
  precision_lassologit
1            0.2105263
```

```
# Test set: recall
summarize(boston_test,
          recall_logit = sum(pred_logit * newtax) / sum(newtax),
          recall_lassologit = sum(pred_lassologit * newtax) / sum(newtax))
```

```
  recall_logit recall_lassologit
1          0.9               0.8
```

Exercise: Programming ML in R – Plotting ROC Curve

```
roc(newtax ~ prob_lassologit, boston_test, plot = TRUE)
```



Wrap Up

Have a great weekend!

I will send out a poll to determine the best time to do final exam review.