Lab 5

Adeline Casali

2024-02-15

Data preprocessing, neural network model training, and ROC + Calibration curve creation

# Loading packages and data  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(reticulate)  
library(tensorflow)

##   
## Attaching package: 'tensorflow'

## The following object is masked from 'package:caret':  
##   
## train

library(keras)  
library(MESS)  
data <- read.csv("lab\_5\_data.csv")  
  
# Create testing and training sets  
training\_ind <- createDataPartition(data$lodgepole\_pine,   
 p = 0.75,   
 list = FALSE,   
 times = 1)  
training\_set <- data[training\_ind, ]  
test\_set <- data[-training\_ind, ]  
  
# Assessing, grouping, and factoring categorical variables  
unique(training\_set$wilderness\_area)

## [1] "wilderness\_area\_1" "wilderness\_area\_3" "wilderness\_area\_4"  
## [4] "wilderness\_area\_2"

unique(training\_set$soil\_type)

## [1] "soil\_type\_18" "soil\_type\_30" "soil\_type\_12" "soil\_type\_29" "soil\_type\_20"  
## [6] "soil\_type\_23" "soil\_type\_24" "soil\_type\_22" "soil\_type\_10" "soil\_type\_11"  
## [11] "soil\_type\_33" "soil\_type\_39" "soil\_type\_13" "soil\_type\_31" "soil\_type\_2"   
## [16] "soil\_type\_5" "soil\_type\_17" "soil\_type\_1" "soil\_type\_32" "soil\_type\_3"   
## [21] "soil\_type\_6" "soil\_type\_16" "soil\_type\_40" "soil\_type\_35" "soil\_type\_38"  
## [26] "soil\_type\_27" "soil\_type\_25" "soil\_type\_4" "soil\_type\_19" "soil\_type\_8"   
## [31] "soil\_type\_9" "soil\_type\_28" "soil\_type\_34" "soil\_type\_14" "soil\_type\_37"  
## [36] "soil\_type\_21" "soil\_type\_36" "soil\_type\_26"

top\_20\_soil\_types <- training\_set %>%   
 group\_by(soil\_type) %>%   
 summarize(count = n()) %>%   
 arrange(desc(count)) %>%  
 select(soil\_type) %>%   
 top\_n(20)

## Selecting by soil\_type

training\_set$soil\_type <- ifelse(training\_set$soil\_type %in% top\_20\_soil\_types$soil\_type,   
 training\_set$soil\_type,   
 "other")  
  
training\_set$wilderness\_area <- factor(training\_set$wilderness\_area)  
training\_set$soil\_type <- factor(training\_set$soil\_type)  
  
class(training\_set$wilderness\_area)

## [1] "factor"

class(training\_set$soil\_type)

## [1] "factor"

levels(training\_set$wilderness\_area)

## [1] "wilderness\_area\_1" "wilderness\_area\_2" "wilderness\_area\_3"  
## [4] "wilderness\_area\_4"

levels(training\_set$soil\_type)

## [1] "other" "soil\_type\_27" "soil\_type\_28" "soil\_type\_29" "soil\_type\_3"   
## [6] "soil\_type\_30" "soil\_type\_31" "soil\_type\_32" "soil\_type\_33" "soil\_type\_34"  
## [11] "soil\_type\_35" "soil\_type\_36" "soil\_type\_37" "soil\_type\_38" "soil\_type\_39"  
## [16] "soil\_type\_4" "soil\_type\_40" "soil\_type\_5" "soil\_type\_6" "soil\_type\_8"   
## [21] "soil\_type\_9"

# One-hot encoding the training set  
onehot\_encoder <- dummyVars(~ wilderness\_area + soil\_type,   
 training\_set[, c("wilderness\_area", "soil\_type")],   
 levelsOnly = TRUE,   
 fullRank = TRUE)  
  
onehot\_enc\_training <- predict(onehot\_encoder,   
 training\_set[, c("wilderness\_area", "soil\_type")])  
training\_set <- cbind(training\_set, onehot\_enc\_training)  
  
# One-hot encoding the test set  
test\_set$soil\_type <- ifelse(test\_set$soil\_type %in% top\_20\_soil\_types$soil\_type,   
 test\_set$soil\_type,   
 "other")  
  
test\_set$wilderness\_area <- factor(test\_set$wilderness\_area)  
test\_set$soil\_type <- factor(test\_set$soil\_type)  
  
onehot\_enc\_test <- predict(onehot\_encoder, test\_set[, c("wilderness\_area", "soil\_type")])  
test\_set <- cbind(test\_set, onehot\_enc\_test)  
  
# Scaling test and training sets  
test\_set[, -c(11:13)] <- scale(test\_set[, -c(11:13)],   
 center = apply(training\_set[, -c(11:13)], 2, mean),   
 scale = apply(training\_set[, -c(11:13)], 2, sd))  
training\_set[, -c(11:13)] <- scale(training\_set[, -c(11:13)])  
  
# Convert data sets to tensors  
training\_features <- array(data = unlist(training\_set[, -c(11:13)]),   
 dim = c(nrow(training\_set), 33))  
training\_labels <- array(data = unlist(training\_set[, 13]),   
 dim = c(nrow(training\_set)))  
  
test\_features <- array(data = unlist(test\_set[, -c(11:13)]),   
 dim = c(nrow(test\_set), 33))  
test\_labels <- array(data = unlist(test\_set[, 13]),   
 dim = c(nrow(test\_set)))  
  
# Building the model  
model <- keras\_model\_sequential(list(  
 layer\_dense(units = 50, activation = "relu"),   
 layer\_dense(units = 25, activation = "relu"),   
 layer\_dense(units = 1, activation = "sigmoid")  
))  
compile(model,   
 optimizer = "rmsprop",   
 loss = "binary\_crossentropy",   
 metrics = "accuracy")  
  
# Training the model  
history <- fit(model, training\_features, training\_labels,   
 epochs = 40, batch\_size = 512, validation\_split = 0.33)

## Epoch 1/40  
## 9/9 - 1s - loss: 0.8179 - accuracy: 0.5444 - val\_loss: 0.7006 - val\_accuracy: 0.5663 - 1s/epoch - 118ms/step  
## Epoch 2/40  
## 9/9 - 0s - loss: 0.6731 - accuracy: 0.6584 - val\_loss: 0.6889 - val\_accuracy: 0.5968 - 127ms/epoch - 14ms/step  
## Epoch 3/40  
## 9/9 - 0s - loss: 0.6178 - accuracy: 0.6979 - val\_loss: 0.6891 - val\_accuracy: 0.5955 - 113ms/epoch - 13ms/step  
## Epoch 4/40  
## 9/9 - 0s - loss: 0.5865 - accuracy: 0.7075 - val\_loss: 0.6939 - val\_accuracy: 0.5950 - 114ms/epoch - 13ms/step  
## Epoch 5/40  
## 9/9 - 0s - loss: 0.5681 - accuracy: 0.7173 - val\_loss: 0.6984 - val\_accuracy: 0.5913 - 111ms/epoch - 12ms/step  
## Epoch 6/40  
## 9/9 - 0s - loss: 0.5555 - accuracy: 0.7296 - val\_loss: 0.6949 - val\_accuracy: 0.5964 - 120ms/epoch - 13ms/step  
## Epoch 7/40  
## 9/9 - 0s - loss: 0.5469 - accuracy: 0.7474 - val\_loss: 0.6960 - val\_accuracy: 0.6033 - 112ms/epoch - 12ms/step  
## Epoch 8/40  
## 9/9 - 0s - loss: 0.5405 - accuracy: 0.7497 - val\_loss: 0.6859 - val\_accuracy: 0.6172 - 136ms/epoch - 15ms/step  
## Epoch 9/40  
## 9/9 - 0s - loss: 0.5351 - accuracy: 0.7538 - val\_loss: 0.6893 - val\_accuracy: 0.6163 - 115ms/epoch - 13ms/step  
## Epoch 10/40  
## 9/9 - 0s - loss: 0.5307 - accuracy: 0.7591 - val\_loss: 0.6728 - val\_accuracy: 0.6335 - 112ms/epoch - 12ms/step  
## Epoch 11/40  
## 9/9 - 0s - loss: 0.5262 - accuracy: 0.7607 - val\_loss: 0.6835 - val\_accuracy: 0.6321 - 112ms/epoch - 12ms/step  
## Epoch 12/40  
## 9/9 - 0s - loss: 0.5223 - accuracy: 0.7673 - val\_loss: 0.6779 - val\_accuracy: 0.6409 - 112ms/epoch - 12ms/step  
## Epoch 13/40  
## 9/9 - 0s - loss: 0.5192 - accuracy: 0.7675 - val\_loss: 0.6758 - val\_accuracy: 0.6515 - 113ms/epoch - 13ms/step  
## Epoch 14/40  
## 9/9 - 0s - loss: 0.5165 - accuracy: 0.7700 - val\_loss: 0.6796 - val\_accuracy: 0.6492 - 113ms/epoch - 13ms/step  
## Epoch 15/40  
## 9/9 - 0s - loss: 0.5154 - accuracy: 0.7700 - val\_loss: 0.6702 - val\_accuracy: 0.6525 - 113ms/epoch - 13ms/step  
## Epoch 16/40  
## 9/9 - 0s - loss: 0.5144 - accuracy: 0.7694 - val\_loss: 0.6688 - val\_accuracy: 0.6538 - 115ms/epoch - 13ms/step  
## Epoch 17/40  
## 9/9 - 0s - loss: 0.5137 - accuracy: 0.7712 - val\_loss: 0.6680 - val\_accuracy: 0.6534 - 112ms/epoch - 12ms/step  
## Epoch 18/40  
## 9/9 - 0s - loss: 0.5118 - accuracy: 0.7714 - val\_loss: 0.6697 - val\_accuracy: 0.6548 - 115ms/epoch - 13ms/step  
## Epoch 19/40  
## 9/9 - 0s - loss: 0.5115 - accuracy: 0.7741 - val\_loss: 0.6693 - val\_accuracy: 0.6589 - 112ms/epoch - 12ms/step  
## Epoch 20/40  
## 9/9 - 0s - loss: 0.5112 - accuracy: 0.7730 - val\_loss: 0.6670 - val\_accuracy: 0.6557 - 113ms/epoch - 13ms/step  
## Epoch 21/40  
## 9/9 - 0s - loss: 0.5102 - accuracy: 0.7771 - val\_loss: 0.6689 - val\_accuracy: 0.6585 - 112ms/epoch - 12ms/step  
## Epoch 22/40  
## 9/9 - 0s - loss: 0.5099 - accuracy: 0.7719 - val\_loss: 0.6648 - val\_accuracy: 0.6594 - 113ms/epoch - 13ms/step  
## Epoch 23/40  
## 9/9 - 0s - loss: 0.5102 - accuracy: 0.7744 - val\_loss: 0.6609 - val\_accuracy: 0.6640 - 112ms/epoch - 12ms/step  
## Epoch 24/40  
## 9/9 - 0s - loss: 0.5106 - accuracy: 0.7760 - val\_loss: 0.6674 - val\_accuracy: 0.6640 - 113ms/epoch - 13ms/step  
## Epoch 25/40  
## 9/9 - 0s - loss: 0.5090 - accuracy: 0.7726 - val\_loss: 0.6684 - val\_accuracy: 0.6645 - 113ms/epoch - 13ms/step  
## Epoch 26/40  
## 9/9 - 0s - loss: 0.5097 - accuracy: 0.7771 - val\_loss: 0.6713 - val\_accuracy: 0.6640 - 112ms/epoch - 12ms/step  
## Epoch 27/40  
## 9/9 - 0s - loss: 0.5097 - accuracy: 0.7751 - val\_loss: 0.6672 - val\_accuracy: 0.6640 - 113ms/epoch - 13ms/step  
## Epoch 28/40  
## 9/9 - 0s - loss: 0.5091 - accuracy: 0.7751 - val\_loss: 0.6789 - val\_accuracy: 0.6627 - 114ms/epoch - 13ms/step  
## Epoch 29/40  
## 9/9 - 0s - loss: 0.5100 - accuracy: 0.7741 - val\_loss: 0.6657 - val\_accuracy: 0.6742 - 113ms/epoch - 13ms/step  
## Epoch 30/40  
## 9/9 - 0s - loss: 0.5097 - accuracy: 0.7741 - val\_loss: 0.6744 - val\_accuracy: 0.6636 - 114ms/epoch - 13ms/step  
## Epoch 31/40  
## 9/9 - 0s - loss: 0.5090 - accuracy: 0.7780 - val\_loss: 0.6753 - val\_accuracy: 0.6562 - 112ms/epoch - 12ms/step  
## Epoch 32/40  
## 9/9 - 0s - loss: 0.5100 - accuracy: 0.7728 - val\_loss: 0.6774 - val\_accuracy: 0.6474 - 117ms/epoch - 13ms/step  
## Epoch 33/40  
## 9/9 - 0s - loss: 0.5092 - accuracy: 0.7773 - val\_loss: 0.6821 - val\_accuracy: 0.6446 - 118ms/epoch - 13ms/step  
## Epoch 34/40  
## 9/9 - 0s - loss: 0.5085 - accuracy: 0.7780 - val\_loss: 0.6766 - val\_accuracy: 0.6580 - 114ms/epoch - 13ms/step  
## Epoch 35/40  
## 9/9 - 0s - loss: 0.5097 - accuracy: 0.7723 - val\_loss: 0.6781 - val\_accuracy: 0.6455 - 113ms/epoch - 13ms/step  
## Epoch 36/40  
## 9/9 - 0s - loss: 0.5084 - accuracy: 0.7773 - val\_loss: 0.6794 - val\_accuracy: 0.6515 - 113ms/epoch - 13ms/step  
## Epoch 37/40  
## 9/9 - 0s - loss: 0.5093 - accuracy: 0.7755 - val\_loss: 0.6905 - val\_accuracy: 0.6372 - 112ms/epoch - 12ms/step  
## Epoch 38/40  
## 9/9 - 0s - loss: 0.5100 - accuracy: 0.7723 - val\_loss: 0.6841 - val\_accuracy: 0.6418 - 113ms/epoch - 13ms/step  
## Epoch 39/40  
## 9/9 - 0s - loss: 0.5091 - accuracy: 0.7757 - val\_loss: 0.6850 - val\_accuracy: 0.6423 - 120ms/epoch - 13ms/step  
## Epoch 40/40  
## 9/9 - 0s - loss: 0.5093 - accuracy: 0.7760 - val\_loss: 0.6944 - val\_accuracy: 0.6335 - 111ms/epoch - 12ms/step

# Calculating probabilities  
predictions <- predict(model, test\_features)

## 69/69 - 0s - 154ms/epoch - 2ms/step

test\_set$p\_prob <- predictions[, 1]  
head(predictions, 10)

## [,1]  
## [1,] 0.573110640  
## [2,] 0.736599207  
## [3,] 0.860034883  
## [4,] 0.717008770  
## [5,] 0.001818147  
## [6,] 0.245366052  
## [7,] 0.668084383  
## [8,] 0.174939141  
## [9,] 0.145607367  
## [10,] 0.008559200

# Making predictions and calculating fpr and tpr rates at 0.5 threshold  
over\_threshold <- test\_set[test\_set$p\_prob >= 0.5, ]  
fpr <- sum(over\_threshold$lodgepole\_pine==0)/sum(test\_set$lodgepole\_pine==0)  
fpr

## [1] 0.3268032

tpr <- sum(over\_threshold$lodgepole\_pine==1)/sum(test\_set$lodgepole\_pine==1)  
tpr

## [1] 0.8162879

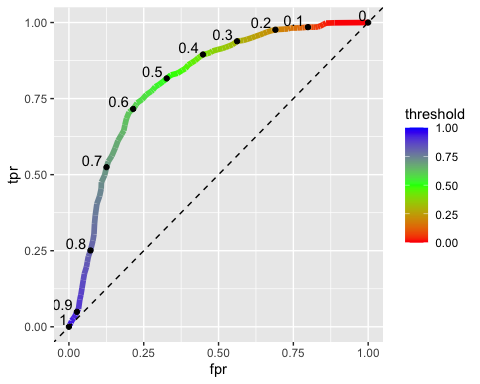
# fpr and tpr at 0.75 threshold  
over\_threshold <- test\_set[test\_set$p\_prob >= 0.75, ]  
fpr <- sum(over\_threshold$lodgepole\_pine==0)/sum(test\_set$lodgepole\_pine==0)  
fpr

## [1] 0.09349955

tpr <- sum(over\_threshold$lodgepole\_pine==1)/sum(test\_set$lodgepole\_pine==1)  
tpr

## [1] 0.4043561

# Plotting ROC curve  
roc\_data <- data.frame(threshold = seq(1, 0, -0.01), fpr = 0, tpr = 0)  
for (i in roc\_data$threshold) {  
 over\_threshold <- test\_set[test\_set$p\_prob >= i, ]  
 fpr <- sum(over\_threshold$lodgepole\_pine==0)/sum(test\_set$lodgepole\_pine==0)  
 roc\_data[roc\_data$threshold==i, "fpr"] <- fpr  
 tpr <- sum(over\_threshold$lodgepole\_pine==1)/sum(test\_set$lodgepole\_pine==1)  
 roc\_data[roc\_data$threshold==i, "tpr"] <- tpr  
}  
ggplot() +   
 geom\_line(data = roc\_data,   
 aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +   
 scale\_color\_gradientn(colors = rainbow(3)) +   
 geom\_abline(intercept = 0, slope = 1, lty = 2) +   
 geom\_point(data = roc\_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +   
 geom\_text(data = roc\_data[seq(1, 101, 10), ],   
 aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



# Calculating the AUC  
auc <- auc(x = roc\_data$fpr, y = roc\_data$tpr, type = "spline")

## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique  
## 'x' values

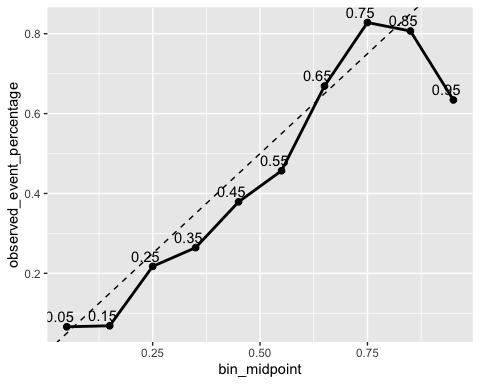
auc

## [1] 0.808372

# Creating a calibration curve  
in\_interval <- test\_set[test\_set$p\_prob >= 0.7 & test\_set$p\_prob <= 0.8, ]  
nrow(in\_interval[in\_interval$lodgepole\_pine==1, ])/nrow(in\_interval)

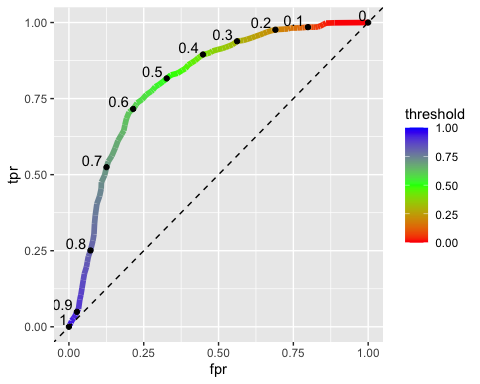
## [1] 0.8280802

calibration\_data <- data.frame(bin\_midpoint=seq(0.05,0.95,0.1),  
 observed\_event\_percentage=0)  
for (i in seq(0.05,0.95,0.1)) {  
 in\_interval <- test\_set[test\_set$p\_prob >= (i-0.05) & test\_set$p\_prob <= (i+0.05), ]  
 oep <- nrow(in\_interval[in\_interval$lodgepole\_pine==1, ])/nrow(in\_interval)  
 calibration\_data[calibration\_data$bin\_midpoint==i, "observed\_event\_percentage"] <- oep  
}  
ggplot(data = calibration\_data, aes(x = bin\_midpoint, y = observed\_event\_percentage)) +  
 geom\_line(linewidth = 1) +  
 geom\_abline(intercept = 0, slope = 1, lty = 2) +  
 geom\_point(size = 2) +  
 geom\_text(aes(label = bin\_midpoint), hjust = 0.75, vjust = -0.5)



1. In the ROC curve above, the tpr and fpr associated with the threshold value of 0.3 are 0.938447 and 0.5618878, respectively.

ggplot() +   
 geom\_line(data = roc\_data,   
 aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +   
 scale\_color\_gradientn(colors = rainbow(3)) +   
 geom\_abline(intercept = 0, slope = 1, lty = 2) +   
 geom\_point(data = roc\_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +   
 geom\_text(data = roc\_data[seq(1, 101, 10), ],   
 aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



over\_threshold <- test\_set[test\_set$p\_prob >= 0.3, ]  
fpr <- sum(over\_threshold$lodgepole\_pine==0)/sum(test\_set$lodgepole\_pine==0)  
fpr

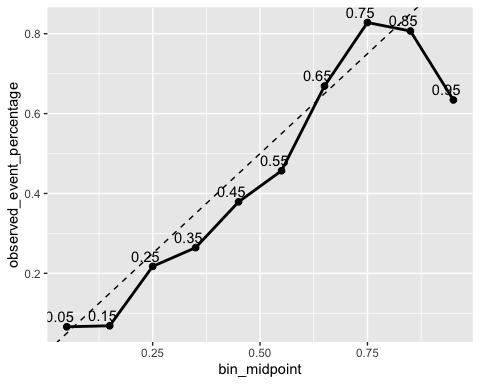
## [1] 0.5618878

tpr <- sum(over\_threshold$lodgepole\_pine==1)/sum(test\_set$lodgepole\_pine==1)  
tpr

## [1] 0.938447

1. In this calibration curve, the predicted probabilities in the interval (0.2, 0.3) are over-confident probabilities, as the point associated with the mid-point (0.25) lies slightly below the diagonal line.

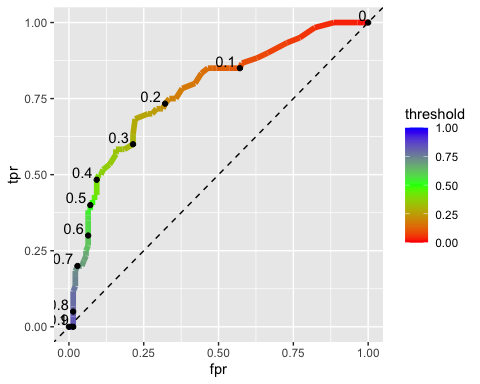
ggplot(data = calibration\_data, aes(x = bin\_midpoint, y = observed\_event\_percentage)) +  
 geom\_line(linewidth = 1) +  
 geom\_abline(intercept = 0, slope = 1, lty = 2) +  
 geom\_point(size = 2) +  
 geom\_text(aes(label = bin\_midpoint), hjust = 0.75, vjust = -0.5)



1. Below are the ROC and calibration curves.

library(AppliedPredictiveModeling)  
data("logisticCreditPredictions")  
lcp <- logisticCreditPredictions # only do this to shorten the name  
#### ROC curve  
roc\_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)  
for (i in roc\_data$threshold) {  
over\_threshold <- lcp[lcp$Bad >= i, ]  
fpr <- sum(over\_threshold$obs=="Good")/sum(lcp$obs=="Good")  
roc\_data[roc\_data$threshold==i, "fpr"] <- fpr  
tpr <- sum(over\_threshold$obs=="Bad")/sum(lcp$obs=="Bad")  
roc\_data[roc\_data$threshold==i, "tpr"] <- tpr  
}  
ggplot() +  
geom\_line(data = roc\_data, aes(x = fpr, y = tpr, color = threshold), size = 2) +  
scale\_color\_gradientn(colors = rainbow(3)) +  
geom\_abline(intercept = 0, slope = 1, lty = 2) +  
geom\_point(data = roc\_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +  
geom\_text(data = roc\_data[seq(1, 101, 10), ],  
aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
## ℹ Please use `linewidth` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



### calibration curve  
calibration\_data <- data.frame(bin\_midpoint=seq(0.05,0.95,0.1),  
observed\_event\_percentage=0)  
for (i in seq(0.05,0.95,0.1)) {  
7  
in\_interval <- lcp[lcp$Bad >= (i-0.05) & lcp$Bad <= (i+0.05), ]  
temp <- nrow(in\_interval[in\_interval$obs=="Bad", ])/nrow(in\_interval)  
calibration\_data[calibration\_data$bin\_midpoint==i, "observed\_event\_percentage"] <- temp  
}  
ggplot(data = calibration\_data, aes(x = bin\_midpoint, y = observed\_event\_percentage)) +  
geom\_line(linewidth = 1) +  
geom\_abline(intercept = 0, slope = 1, lty = 2) +  
geom\_point(size = 2) +  
geom\_text(aes(label = bin\_midpoint), hjust = 0.75, vjust = -0.5)

