Logistic Regression and Regularization

Applied Machine Learning for Educational Data Science

true

10/26/2021

Contents

[Updated: Mon, Nov 01, 2021 - 09:02:07]

1. Overview of the Logistic Regression

Logistic regression is a type of model that can be used to predict a binary outcome variable. Linear regression and logistic regression are indeed members of the same family of models called *generalized linear models*. While linear regression can also technically be used to predict a binary outcome, the bounded nature of a binary outcome, [0,1], makes the linear regression solution suboptimal. Logistic regression is a more appropriate model and takes the bounded nature of the binary outcome into account when making predictions.

The binary outcomes can be coded in a variety of ways in the data such as 0 vs 1, True vs False, Yes vs. No, Success vs. Failure. The rest of the notes, it is assumed that the category of interest to predict is represented by 1s in the data.

The notes in this section will first introduce a suboptimal solution to predict a binary outcome by fitting a linear probability model using linear regression and discuss the limitations of this approach. Then, the logistic regression model and its estimation will be demonstrated. Finally, different regularization approaches for the logistic regression will be discussed.

Throughout these notes, we will use the Recidivism dataset from the NIJ competition to discuss different aspects of logistic regression and demonstrations. This data and variables in this data were discussed in detail in Lecture 1a and Lecture 2a. The outcome of interest to predict in this dataset is whether or not an individual will be recidivated in the second year after initial release. In order to make demonstrations easier, I randomly sample 20 observations from this data. Six observations in this data have a value of 1 for the outcome (recidivated) while 14 observations have a value of 0 (not recidivated).

```
# Download the random sample of 20 observations from the recidivism dataset

recidivism_sub <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/
                           header=TRUE)

# Outcome variable

table(recidivism_sub$Recidivism_Arrest_Year2)
```

```
 0  1
14  6
```

1.1. Linear Probability Model

Linear probability model is just fitting a typical regression model to a binary outcome. When the outcome is binary, the predictions from a linear regression model can be considered as probability of outcome being equal to 1,

$$\hat{Y} = P(Y = 1).$$

Suppose that we want to predict the recidivism in the second year (`Recidivism_Arrest_Year2`) by using the number of dependents they have. Then, we could fit this using the `lm` function.

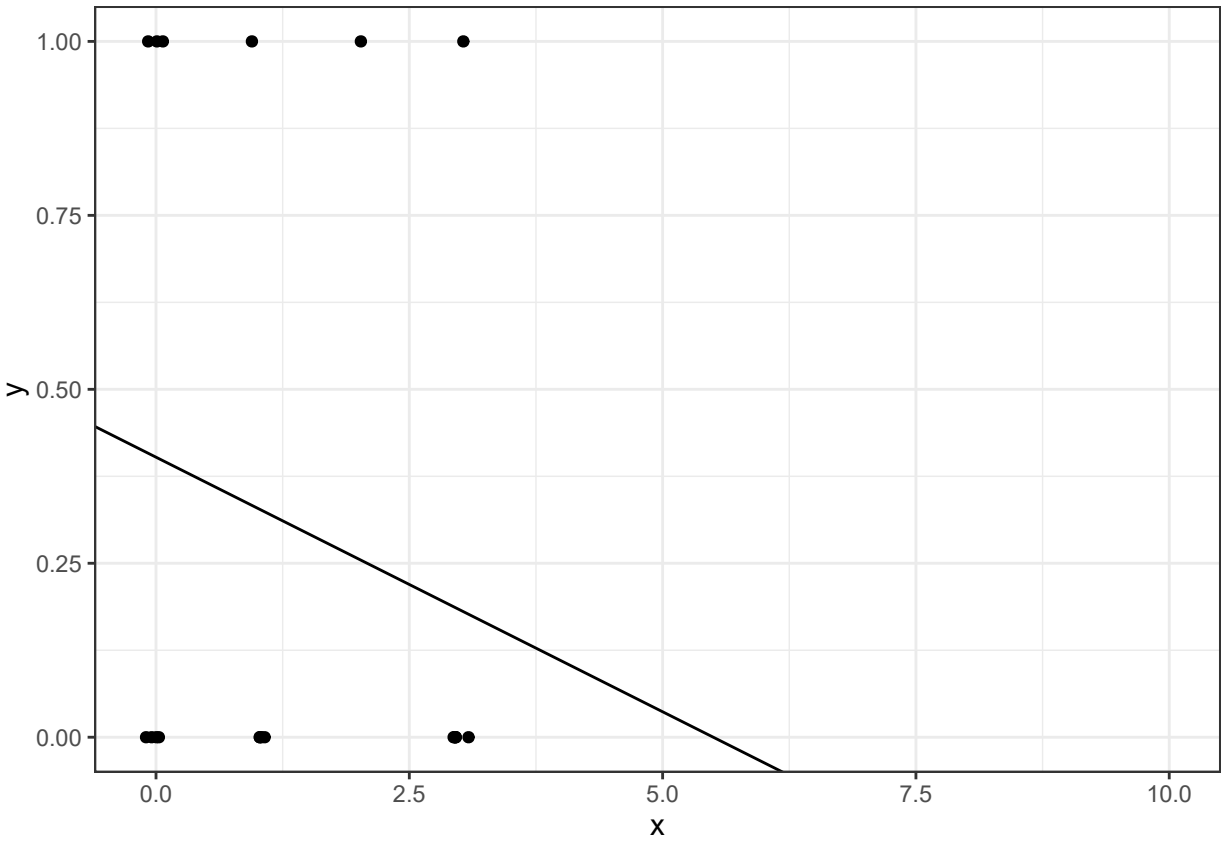
```
mod <- lm(Recidivism_Arrest_Year2 ~ 1 + Dependents,
          data = recidivism_sub)

summary(mod)
```

```
Call:
lm(formula = Recidivism_Arrest_Year2 ~ 1 + Dependents, data = recidivism_sub)

Residuals:
    Min      1Q  Median      3Q     Max
-0.4024 -0.3293 -0.1829  0.5976  0.8171

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.40244    0.15665   2.569   0.0193 *
Dependents  -0.07317    0.08256  -0.886   0.3872
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4728 on 18 degrees of freedom
Multiple R-squared:  0.04181,   Adjusted R-squared:  -0.01142
F-statistic: 0.7855 on 1 and 18 DF,  p-value: 0.3872
```

The intercept is 0.402 and the slope for the predictor `Dependents` is -.073. We can interpret the intercept and slope as the following for this example. Note that the predicted values from this model now can be interpreted as probability predictions because the outcome is binary.

- Intercept (0.402): When the number of dependent is equal to 0, the probability of being recidivated in Year 2 is 0.402.
- Slope (-0.073): For every additional dependent (one unit increase in X) the individual has, the probability of being recidivated in Year 2 is reduced by .07.

The intercept and slope still represent the best fitting line to our data, and this fitted line can be shown here.



Now, suppose we want to calculate the model predicted probability of being recidivated in Year 2 for different number of dependents a parolee has. Let's assume that the number of dependents can be any number from 0 to 10. What would be the predicted probability of being recidivated in Year 2 for a parolee with 8 dependents?

```
X <- data.frame(Dependents = 0:10)

predict(mod,newdata = X)
```

```
         1          2          3          4          5          6
 0.40243902  0.32926829  0.25609756  0.18292683  0.10975610  0.03658537
         7          8          9         10         11
-0.03658537 -0.10975610 -0.18292683 -0.25609756 -0.32926829
```

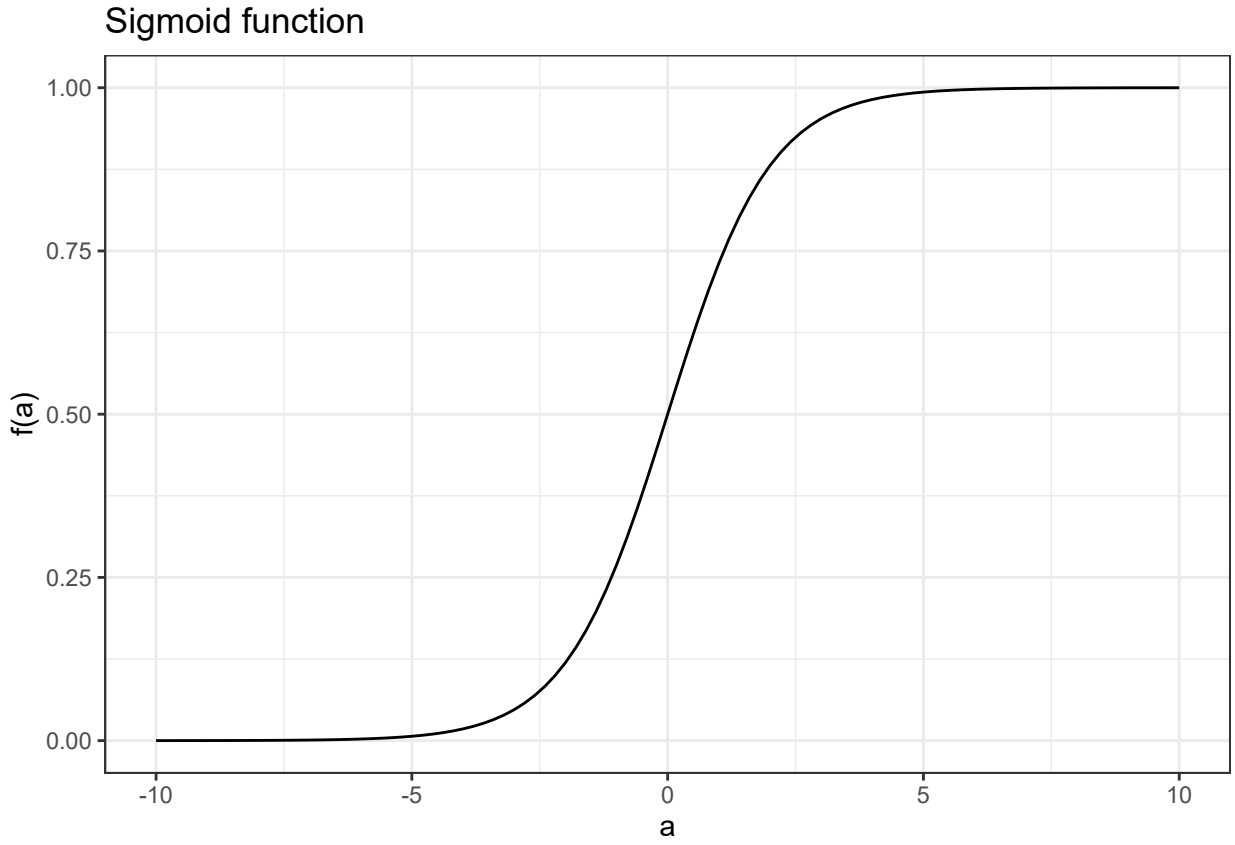
It is not reasonable for a probability prediction to be negative. One of the major issues with using a linear regression to predict a binary outcome using a linear-probability model is that the model predictions can easily be go outside of the boundary [0,1] and yield unreasonable predictions. So, a linear regression model may not necessarily the best tool to predict a binary outcome. We should use a model that respect the boundaries of the outcome variable.

1.2. Description of Logistic Regression Model

In order to overcome the limitations of the linear probability model, we bundle our prediction model in a sigmoid function. Suppose there is real-valued function of a such that

$$f(a) = \frac{e^a}{e^a + 1}.$$

It can be shows than the output of this function is always bounded to be between 0 and 1 regardless of the value of a. Therefore, sigmoid function is an appropriate choice for the logistic regression because it assures that the output is always bounded between 0 and 1.



If we revisit the previous example, we can specify a logistic regression model to predict the probability of being recidivated in Year 2 as the following by using the number of dependents a parolee has as the predictor,

$$P(Y = 1) = \frac{e^{\beta_0 + \beta_1 X}}{e^{\beta_0 + \beta_1 X} + 1}.$$

When the values of predictor variable is entered into the equation, the model output can be directly interpreted as the probability of the binary outcome being equal to 1 (or whatever category and meaning a value of one represents). Then, we assume that the actual outcome follows a binomial distribution with the predicted probability.

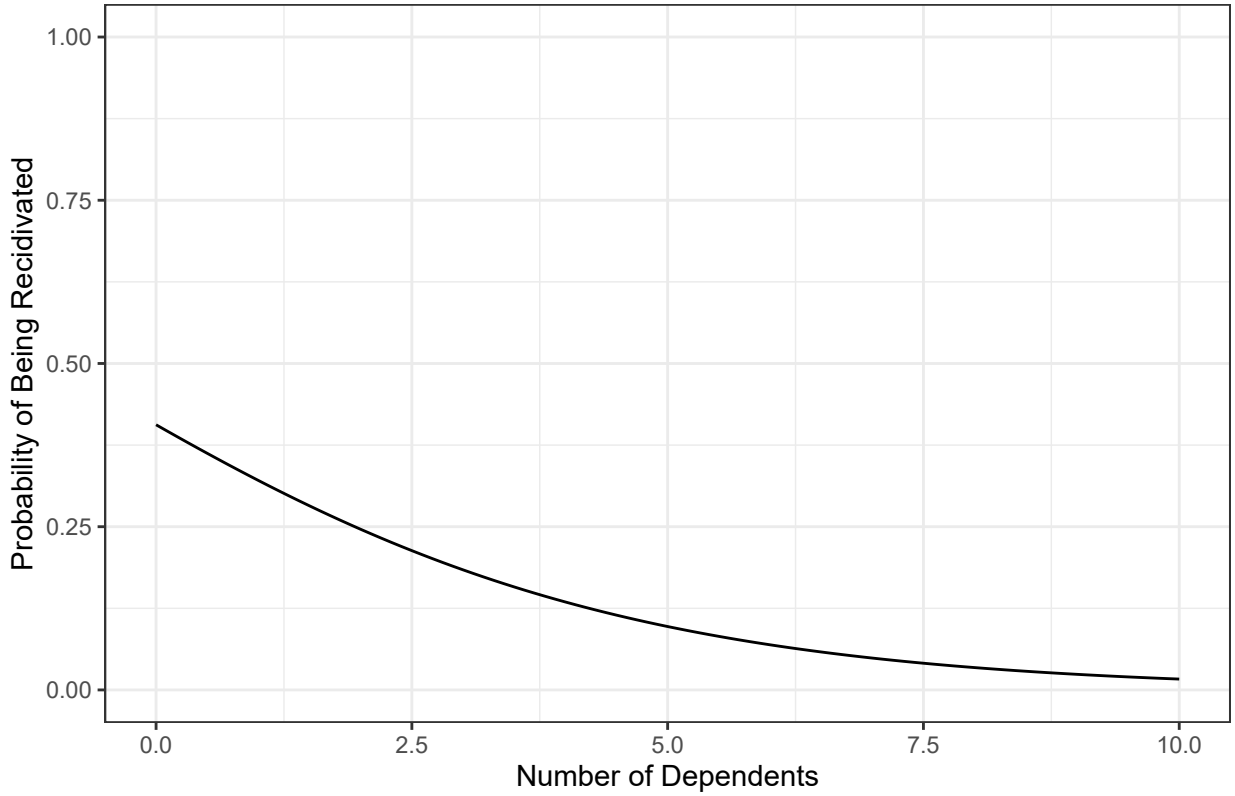
$$P(Y = 1) = p$$
$$Y \sim Binomial(p)$$

Suppose the coefficient estimates of this model are β_0=-0.38 and β_1=-0.37. Then, for instance, we can compute the probability of being recidivated for a parolee with 8 dependents as the following:

$$P(Y=1) = \frac{e^{(-0.38-0.37\times 8)}}{e^{(-0.38-0.37\times 8)+1}} = 0.034.$$

```
b0 = -0.38
b1 = -0.37

x = 0:10

y = exp(b0+b1*x)/(1+exp(b0+b1*x))

data.frame(number.of.dependents = x, probability=y)
```

```
   number.of.dependents probability
1                     0  0.40612690
2                     1  0.32082130
3                     2  0.24601128
4                     3  0.18392173
5                     4  0.13470305
6                     5  0.09708864
7                     6  0.06913842
8                     7  0.04879972
9                     8  0.03422416
10                    9  0.02389269
11                   10  0.01662636
```



In its original form, it is difficult to interpret the parameters of the logistic regression because a one unit increase in the predictor is not anymore linearly related the probability of outcome being equal to 1 due to the nonlinear nature of the sigmoid function. Most common presentation of logistic regression is obtained after a bit of algebraic manipulation to rewrite the model equation. The logistic regression model above can also be specificed as the following without any loss of meaning as they are mathematically equivalent.

$$ln\left[\frac{P(Y=1)}{1-P(Y=1)}\right] = \beta_0 + \beta_1 X.$$

The term on the left side of the equation is known as the **logit**. So, when the outcome is a binary variable, the logit transformation of the probability that the outcome is equal to 1 can be represented as a linear equation. This provides a more straightforward interpretation. For instance, we can know say that when the number of dependents is equal to zero, the predicted logit is equal to -0.38 (intercept), and for every additional dependent the logit decreases by 0.37 (slope).

It is also common to transform the logit to odds when interpreting the parameters. For instance, we can say that when the number of dependents is equal to zero, the odds of being recidivated is 0.68 ($e^{-0.38}$), and for every additional dependent the odds of being recidivated is reduced by 31% ($1 - e^{-0.37}$).

The right side of the equation can be expanded by adding more predictors, adding polynomial terms of the predictors, or adding interactions among predictors. A model with only main effects of P predictors can be written as

$$ln\left[\frac{P(Y=1)}{1-P(Y=1)}\right] = \beta_0 + \sum_{p=1}^{P} \beta_p X_p,$$

and the coefficients can be interpreted as

- β_0: the predicted logit when the values for all the predictor variables in the model are equal to zero. e^{β_0}, the predicted odds of outcome being equal to 1 when the values for all the predictor variables in the model are equal to zero.
- β_p: the change in the predicted logit for one unit increase in X_p when the values for all other predictors in the model are held constant. For every one unit in increase in X_p, the odds of the outcome being equal to 1 is multiplied by e^{β_p} when the values for all other predictors in the model are held constant. In other words, e^{β_p} is odds ratio, the ratio of odds at $\beta_p = a + 1$ to the odds at $\beta_p = a$.

It is important that you get familiar with the three concepts (probability, odds, logit) and how these three are related to each other for interpreting the logistic regression parameters.

NOTE

Sigmoid function is not the only tool to be used for modeling a binary outcome. One can also use the cumulative standard normal distribution function, $\phi(a)$, and the output of $\phi(a)$ is also bounded between 0 and 1. When ϕ is used to to transform the prediction model, this is known as **probit regression** and it serves the same purpose as the logistic regression, which is to predict probability of a binary outcome being equal to 1. However, it is always easier and more pleasant to work with logarithmic functions, and logarithmic functions has nice computational properties. Therefore, logistic regression is more commonly used than the probit regression.

1.3. Model Estimation

1.3.1. The concept of likelihood

It is important to understand the concept of likelihood for estimating the coefficients of a logistic regression model. We will consider a simple example of flipping coins for this. Suppose you flip the same coin 20 times and observe the following outcome. We don't necessarily know whether or not this is a fair coin in which the probability of observing a head or tail is equal to 0.5.

$$\mathbf{Y} = (H, H, H, T, H, H, H, T, H, T)$$

Suppose that we define p as the probability of observing a head when we flip this coin. By definition, the probability of observing a tail is $1 - p$.

$$P(Y = H) = p$$

$$P(Y = T) = 1 - p$$

Then, we can calculate the likelihood of our observations of heads and tails as a function of p.

$$\mathfrak{L}(\mathbf{Y}|p) = p \times p \times p \times (1-p) \times p \times p \times p \times (1-p) \times p \times (1-p)$$

For instance, if we say that this is a fair coin and therefore p is equal to 0.5, then the likelihood of observing 7 heads and 3 tails would be equal to

$$\mathfrak{L}(\mathbf{Y}|p = 0.5) = 0.5 \times 0.5 \times 0.5 \times (1-0.5) \times 0.5 \times 0.5 \times 0.5 \times (1-0.5) \times 0.5 \times (1-0.5) = 0.0009765625$$

On the other hand, another person can say this is probably not a fair coin and the p should be something higher than 0.5. How about 0.65?

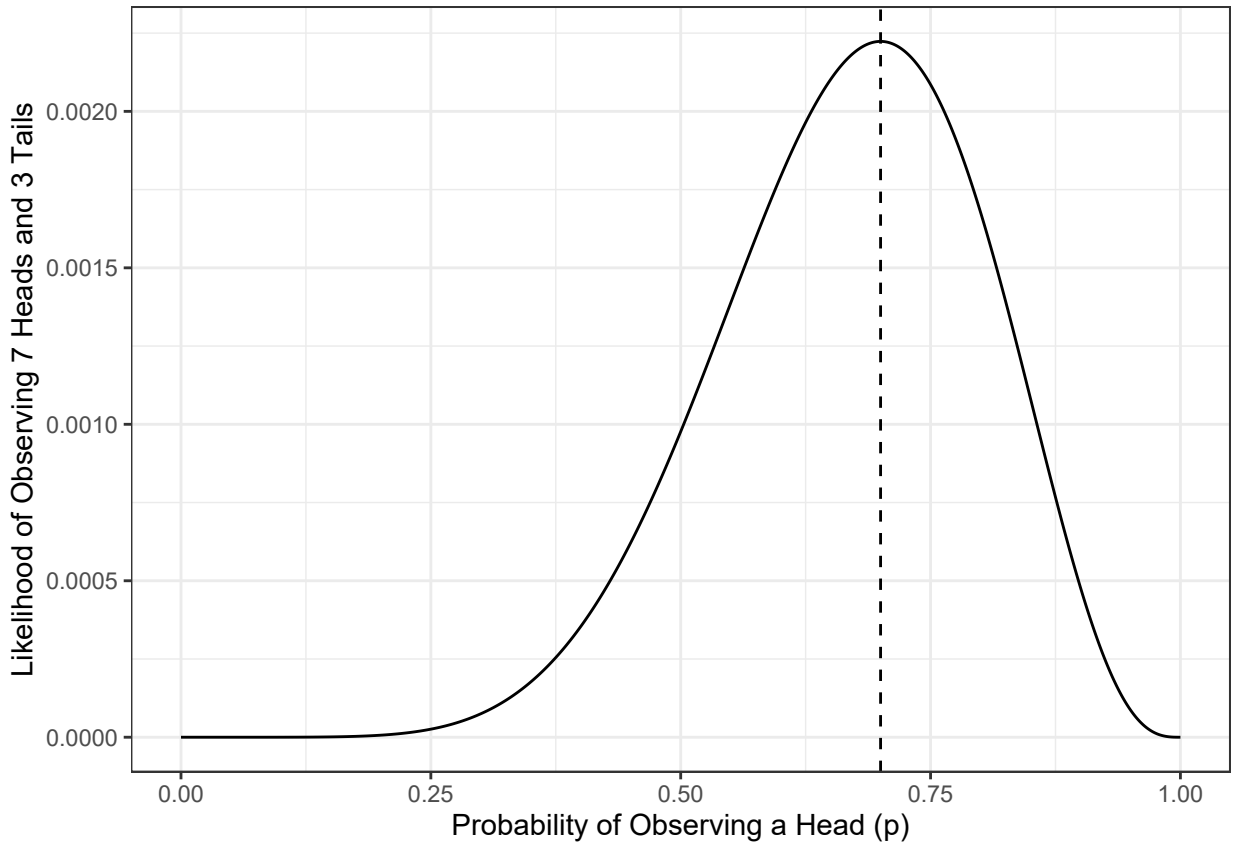
$$\mathfrak{L}(\mathbf{Y}|p = 0.65) = 0.65 \times 0.5 \times 0.65 \times (1-0.65) \times 0.65 \times 0.65 \times 0.65 \times (1-0.65) \times 0.65 \times (1-0.65) = 0.00210183$$

We can say that based on our observation, an estimate of p being equal to 0.65 is more likely than an estimate of p being equal to 0.5. Our observation of 7 heads and 3 tails is more likely if we estimate p as 0.65 rather than 0.5.

1.3.2. Maximum likelihood estimation (MLE)

Then, what would be the best estimate of p given our observed data (7 heads and 3 tails). We can try every possible value of p between 0 and 1, and calculate the likelihood of our data ($\mathbf{Y}$). Then, we can pick the value that makes our data most likely (largest likelihood) to observe as our best estimate. This would be called the maximum likelihood estimate of p given the data we observed.

```
p <- seq(0,1,.001)
L <- p^7*(1-p)^3

ggplot()+
  geom_line(aes(x=p,y=L)) +
  theme_bw() +
  xlab('Probability of Observing a Head (p)')+
  ylab('Likelihood of Observing 7 Heads and 3 Tails')+
  geom_vline(xintercept=p[which.max(L)],lty=2)
```



We can show that the p value that makes the likelihood largest is 0.7, and the likelihood of observing 7 heads and 3 tails is 0.002223566 when p is equal to 0.7. Therefore, the maximum likelihood estimate of probability of observing a head for this particular coin is 0.7 given the 10 observations we have made.

```
L[which.max(L)]
```

```
[1] 0.002223566
```

```
p[which.max(L)]
```

```
[1] 0.7
```

Note that our estimate can change and be updated if we continue collecting more data by flipping the same coin and record our observations.

1.3.3. The concept of loglikelihood

The computation of likelihood requires the multiplication of so many p values, and When you multiply values between 0 and 1, the result gets smaller and smaller. This creates problems when you multiply so many of these small p values due the maximum precision any computer can handle. For instance, you can see what is the minimum number that can be represented in R and meets the requirements of IEEE 754 technical standard.

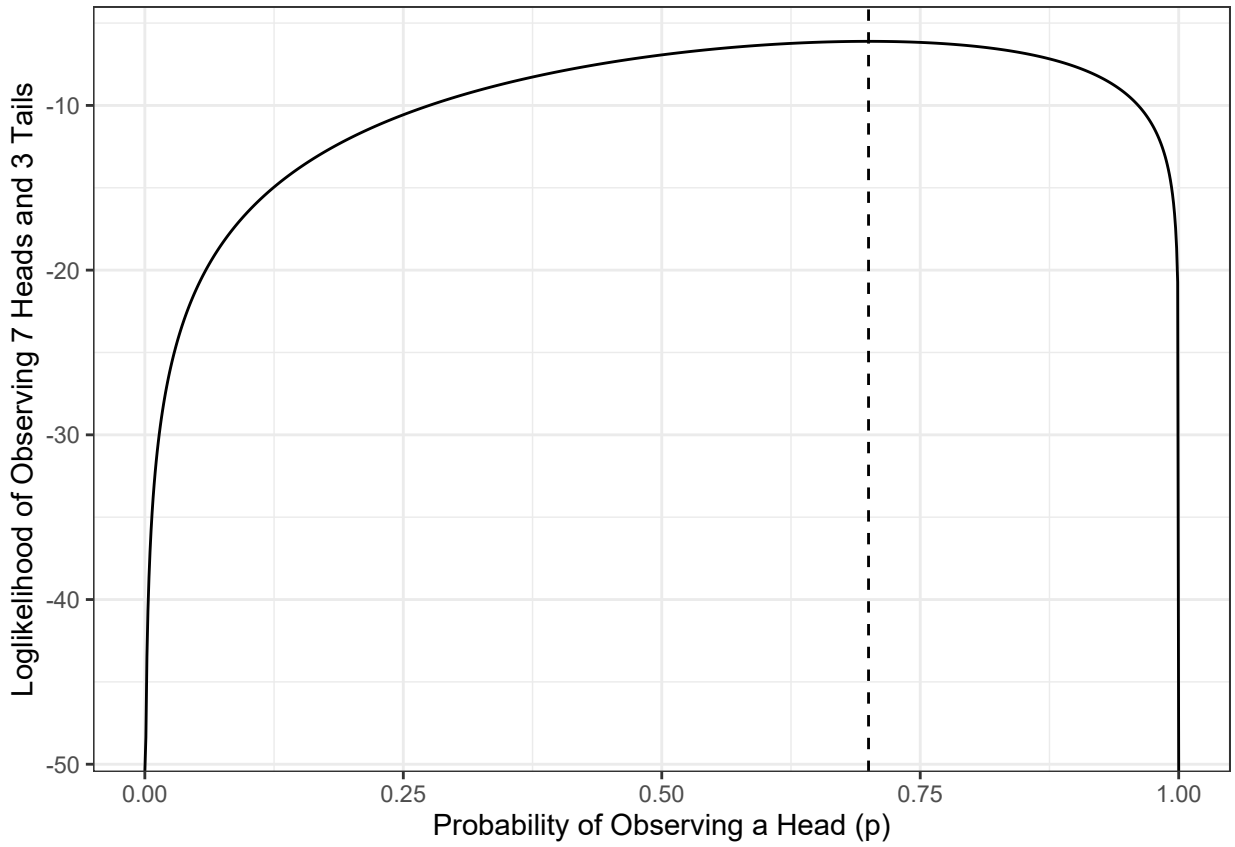
```
.Machine$double.xmin
```

```
[1] 2.225074e-308
```

When you have hundreds of thousands of observations, it is probably not a good idea to directly work with likelihood. Instead, we work with the log of likelihood. This has two main advantages:

- We are less concerned about the precision of small numbers our computer can handle.
- Loglikelihood has nicer mathematical properties to work with for optimization problems (log of product of two numbers is equal to the sum of log of the two numbers).
- The point that maximizes likelihood also the same number that maximizes the loglikelihood, so our end results (MLE estimate) does not care if we use loglikelihood instead of likelihood.

```
p <- seq(0,1,.001)
logL <- log(p)*7 + log(1-p)*3

ggplot()+
  geom_line(aes(x=p,y=logL)) +
  theme_bw() +
  xlab('Probability of Observing a Head (p)')+
  ylab('Loglikelihood of Observing 7 Heads and 3 Tails')+
  geom_vline(xintercept=p[which.max(logL)],lty=2)
```



```
logL[which.max(logL)]
```

```
[1] -6.108643
```

```
p[which.max(logL)]
```

```
[1] 0.7
```

1.3.4. MLE for Logistic Regression coefficients

Now, we can apply these concepts to estimate the logistic regression coefficients. Let's revisit our previous example in which we predict the probability of being recidivated in Year 2 given the number of dependents a parolee has. Our model can be written as the following.

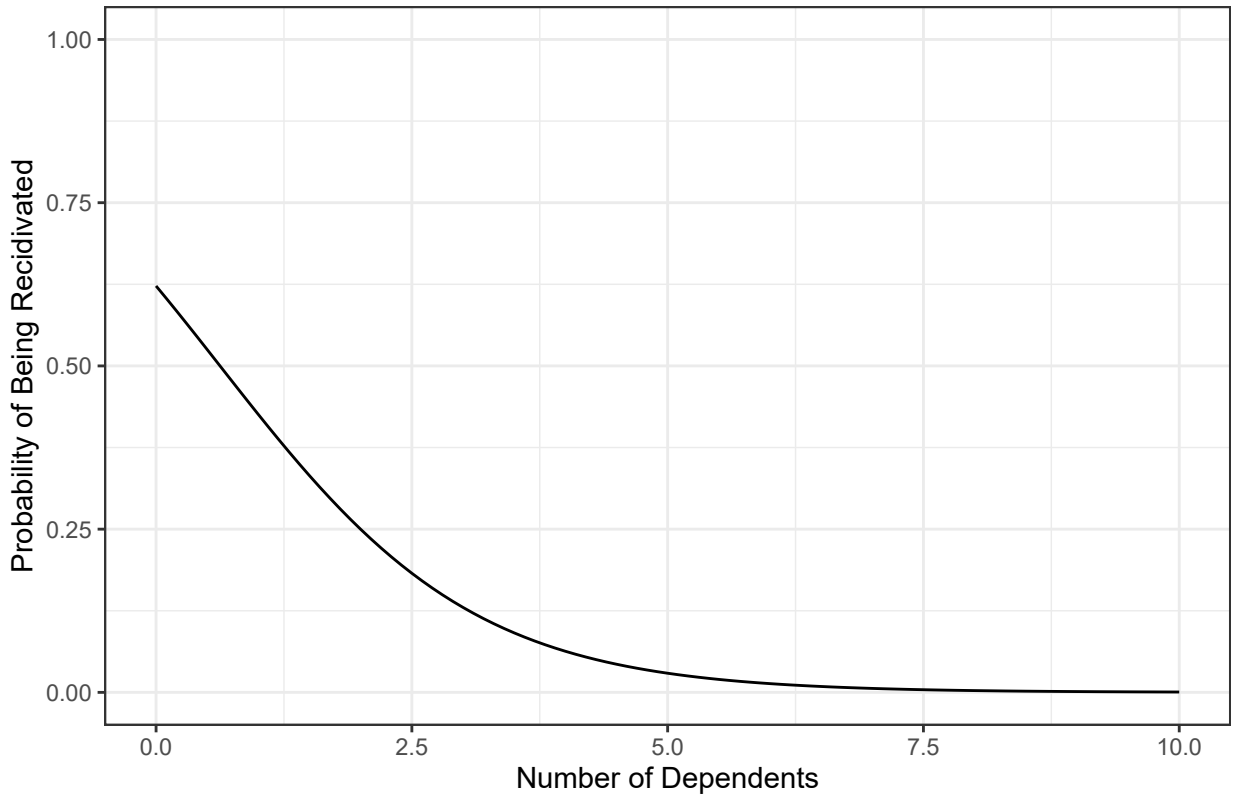
$$ln\left[\frac{P_i(Y=1)}{1-P_i(Y=1)}\right] = \beta_0 + \beta_1 X_i.$$

Note that X and P has a subscript i to indicate that each individual may have a different X value, and therefore each individual will have a different probability. Our observed outcome is a set of 0s and 1s. Remember that there are 6 individuals recidivated (Y=1) and 14 individuals not recidivated (Y=0).

```
recidivism_sub$Recidivism_Arrest_Year2
```

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

Given a set of coefficients, $\{\beta_0, \beta_1\}$, we can calculate the logit for every observation using the model equation, and then transform this logit to a probability,$P_i(Y=1)$. Finally, we can calculate the log of the probability for each obsrvation, and sum them across observations to obtain the loglikelihood of observing this set of observations (14 ones and 6 zeros). Suppose that we have two guesstimates for $\{\beta_0, \beta_1\}$, and they are 0.5 and -0.8, respectively. These coefficients imply the following predicted model.



If these two coefficients are our estimates, how likely would it be to observe the outcome in our data given the number of dependents. The below R code first finds the predicted logit for every single observation assuming that $\beta_0 = 0.5$ and $\beta_1 = -0.8$.

```
b0 = 0.5
b1 = -0.8

x = recidivism_sub$Dependents
y = recidivism_sub$Recidivism_Arrest_Year2

pred_logit <- b0 + b1*x

pred_prob1 <- exp(pred_logit)/(1+exp(pred_logit))
pred_prob0 <- 1 - pred_prob1

data.frame(Dependents      = x,
           Recidivated     = y,
           Prob1 = pred_prob1,
           Prob0 = pred_prob0)
```

```
  Dependents Recidivated     Prob1     Prob0
1          0           1 0.6224593 0.3775407
2          1           0 0.4255575 0.5744425
3          2           1 0.2497399 0.7502601
4          1           0 0.4255575 0.5744425
```

```
5           1           0 0.4255575 0.5744425
6           0           0 0.6224593 0.3775407
7           0           0 0.6224593 0.3775407
8           1           1 0.4255575 0.5744425
9           3           1 0.1301085 0.8698915
10          0           0 0.6224593 0.3775407
11          1           0 0.4255575 0.5744425
12          0           1 0.6224593 0.3775407
13          0           0 0.6224593 0.3775407
14          3           0 0.1301085 0.8698915
15          3           0 0.1301085 0.8698915
16          0           1 0.6224593 0.3775407
17          3           0 0.1301085 0.8698915
18          3           0 0.1301085 0.8698915
19          3           0 0.1301085 0.8698915
20          3           0 0.1301085 0.8698915
```

```
logL <-  y*log(pred_prob1) + (1-y)*log(pred_prob0)

sum(logL)
```

```
[1] -12.65336
```

We can sumamrize this by saying that if our model coefficients are $\beta_0 = 0.5$ and $\beta_1 = -0.8$, then the log of likelihood of observing the outcome in our data would be -12.65.

$$\mathbf{Y} = (1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0)$$

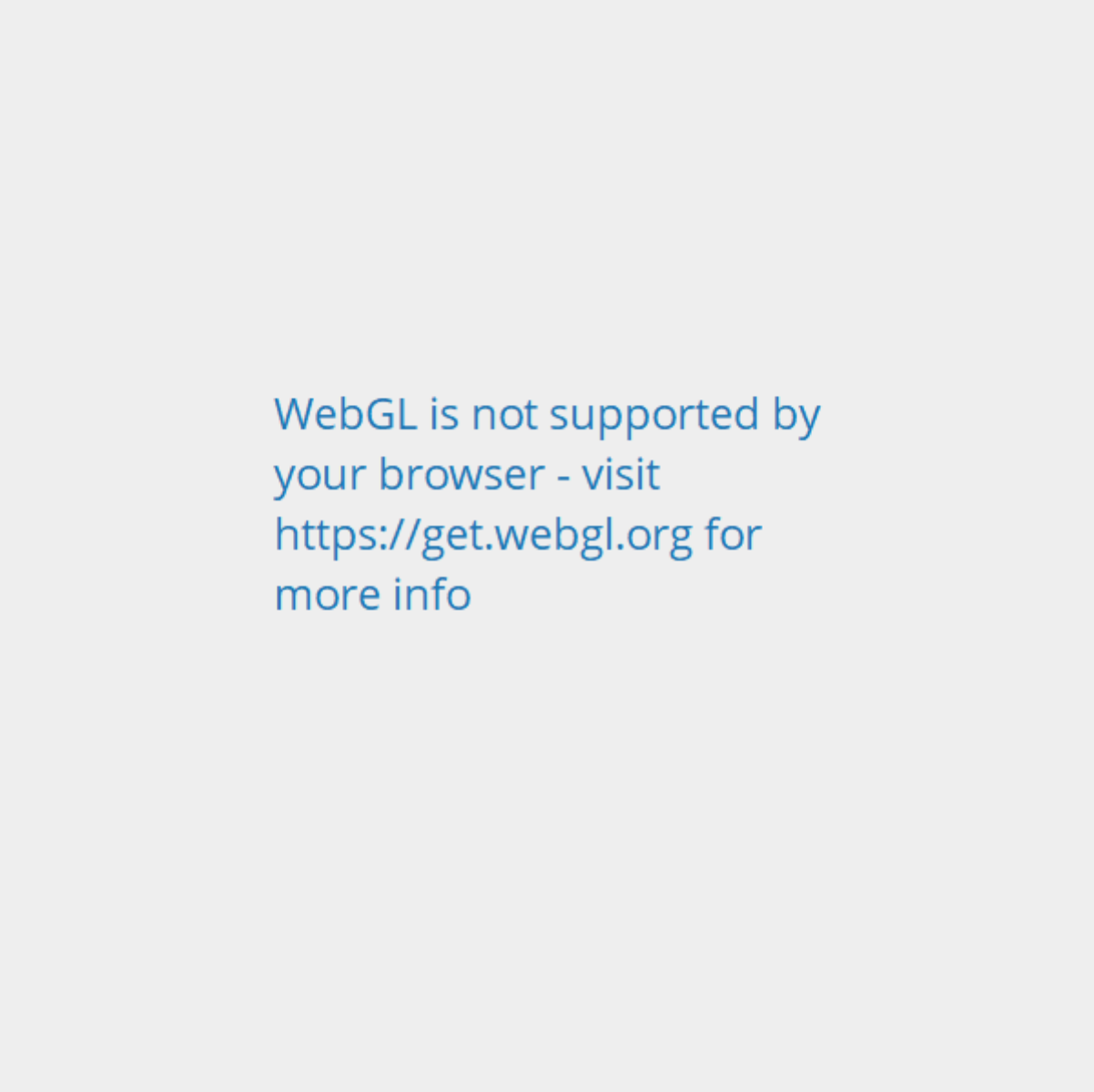
$$\mathfrak{log}\mathfrak{L}(\mathbf{Y}|\beta_0 = 0.5, \beta_1 = -0.8) = -12.65$$

The critical question to ask is whether or not there is another pair of values we can assign to β_0 and β_1 that would provide a higher likelihood of data. If there is, then they would be better estimates for our model. If we can find such a pair with the maximum loglikelihood of data, then they would be our maximum likelihood estimates for the given model.

We can approach this problem in a very crude way to gain some intuition about what Maximum Likelihood Estimation is about. Now, suppose that a reasonable range of values for β_0 is from -1 to 1 and a reasonable range of values for β_1 is also from -1 to 1. Let's think about every possible combinations of values for β_0 and β_1 within these ranges with increments of .01. Then, let's calculate the loglikelihood of data for every possible combination and plot these in a 3D plot as a function of β_0 and β_1.

```
grid    <- expand.grid(b0=seq(-1,1,.01),b1=seq(-1,1,.01))
grid$logL <- NA

for(i in 1:nrow(grid)){

  x = recidivism_sub$Dependents
  y = recidivism_sub$Recidivism_Arrest_Year2

  pred_logit    <- grid[i,]$b0 + grid[i,]$b1*x
  pred_prob1    <- exp(pred_logit)/(1+exp(pred_logit))
  pred_prob0    <- 1 - pred_prob1
```

```
  logL          <- y*log(pred_prob1) + (1-y)*log(pred_prob0)
  grid[i,]$logL <- sum(logL)

  print(i)
}


require(plotly)

plot_ly(grid, x = ~b0, y = ~b1, z = ~logL,
        marker = list(color = ~logL,
                      showscale = FALSE,
                      cmin=min(grid$logL),
                      cmax=max(grid$logL),cauto=F),
        width=600,height=600) %>%
  add_markers()
```



What is the maximum point of this surface? Our crude search indicates that it is -11.78708, and the set of β_0 and β_1 coefficients that make the observed data most likely is -0.38 and -0.37.

```
grid[which.max(grid$logL),]
```

```
         b0    b1      logL
12726 -0.38 -0.37 -11.78708
```

Therefore, given our dataset with 20 observations, our maximum likelihood estimates for the coefficients of the logistic regression model above are -0.38 and -0.37.

$$ln\left[\frac{P_i(Y=1)}{1-P_i(Y=1)}\right] = -0.38 - 0.37 \times X_i.$$

1.3.5. Logistic Loss function

Below is a compact way of writing likelihood and loglikeihood in mathematical notation. For simplification purposes, we write P_i to represent $P_i(Y = 1)$.

$$\mathfrak{L}(\mathbf{Y}|\boldsymbol{\beta}) = \prod_{i=1}^{N} P_i^{y_i} \times (1 - P_i)^{1-y_i}$$

$$\mathfrak{log}\mathfrak{L}(\mathbf{Y}|\boldsymbol{\beta}) = \sum_{i=1}^{N} Y_i \times ln(P_i) + (1 - Y_i) \times ln(1 - P_i)$$

The final equation above, $\mathfrak{log}\mathfrak{L}(\mathbf{Y}|\boldsymbol{\beta})$, is known as the **logistic loss** function. By finding the set of coefficients in a model, $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_P)$, that maximizes this quantity, we obtain the maximum likelihood estimates of the coefficients for the logistic regression model.

Unfortunately, the naive crude search we applied above would be a bad solution when you have a complex model with so many predictors. Another unfortunate thing is that there is no closed form solution (as we had for the linear regression) for the logistic regression. Therefore, the only way to estimate the logistic regression coefficients is to use numerical approximations and computational algorithms to maximize the logistic loss function. Luckily, we have tools available to accomplish this task.

NOTE

Why do we not use least square estimation and minimize the sum of squared residuals when estimating the coefficients of the logistic regression model? We can certainly use the sum of squared residuals as our loss function and minimize it to estimate the coefficients for the logistic regression, just like we did for the linear regression. The complication is that the sum of squared residuals function yields a non-convex surface when the outcome is binary as opposed to a convex surface obtained from the logistic loss function. Non-convex optimization problems are harder than convex optimization problems, and they are more vulnerable in terms of finding sub-optimal solutions (local minima/maxima). Therefore, the logistic loss function and maximizing it is preferred when estimating the coefficients of a logistic regression model.

1.3.6. The `glm` function

The `glm()` function as a part of base `stats` package can be used to estimate the coefficients of the logistic regression. Let's fit the model discussed in the earlier sections using the `glm()` function.

The use of the `glm()` function is very similar to the `lm()` function. The only difference is that we specify the `family='binomial'` argument to fit the logistic regression by maximizing the logistic loss function.

```
mod <- glm(Recidivism_Arrest_Year2 ~ 1 + Dependents,
           data   = recidivism_sub,
           family = 'binomial')

summary(mod)
```

```
Call:
glm(formula = Recidivism_Arrest_Year2 ~ 1 + Dependents, family = "binomial",
    data = recidivism_sub)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.0213  -0.8812  -0.6407   1.3419   1.8355

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.3788     0.6851  -0.553    0.580
Dependents   -0.3668     0.4078  -0.899    0.368

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 24.435  on 19  degrees of freedom
Residual deviance: 23.574  on 18  degrees of freedom
AIC: 27.574

Number of Fisher Scoring iterations: 4
```

In the **Coefficients** table, the numbers under the **Estimate** column are the estimated coefficients for the logistic regression model. The quantity labeled as the **Residual Deviance** in the output is twice the maximized loglikelihood,

$$Deviance = -2 \times \log\mathfrak{L}(\mathbf{Y}|\boldsymbol{\beta}).$$

Notice that the coefficient estimates from the `glm()` function are very close to our crude estimates from a brute-force search in an earlier section (-0.38 and -0.37). From our crude search, we found the maximum loglikelihood as -11.787. So, if we multiply that number by -2, that is equal to 23.574, which is the number reported in this output as **Residual Deviance**.

1.3.7. The `glmnet` function

You can also `glmnet()` function from the `glmnet` package to fit the logistic regression. The advantage of the `glmnet` package is that it provides the option of regularization while fitting the logistic regression. You can set the `alpha=0` and `lambda=0` arguments to obtain the coefficient estimates without any penalty.

```
require(glmnet)

mod <- glmnet(x         = cbind(0,recidivism_sub$Dependents),
              y         = factor(recidivism_sub$Recidivism_Arrest_Year2),
              family    = 'binomial',
              alpha     = 0,
              lambda    = 0,
              intercept = TRUE)

coef(mod)
```

```
3 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept) -0.3787868
V1           .
V2          -0.3668287
```

The `x` argument is the input matrix for predictors, and `y` argument is a vector of binary response outcome. The `glmnet` requires the `y` argument to be a factor with two levels. Note that I defined `x` argument above as `cbind(0,recidivism_sub$Dependents)` because `glmnet` requires the `x` to be a matrix with at least two columns. So, I added a column of zeros to trick the function and force to run it. That column of zeros has zero impact on the estimation.

1.4. Building a Prediction Model for Recidivism

In earlier weeks, we discussed how to process 48 variables in the recidivism data and constructed 165 features for a given individual using a blueprint. Let's read the original data.

```
recidivism <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/mai
```

This dataset has 18,111 rows and 54 columns. Each row represents a parolee. There are 48 predictor variables, one ID variable, four possible outcome variable (Recidivism in Year 1, Recidivism in Year 2, Recidivism in Year 3, and Recidivism within 3 years), and a filter variable to obtain the train and test split as in the original competition. For this lecture, we are interested in predicting Recidivism in Year 2. This dataset has already cleaned by removing the individuals who were recidivated in Year 1 (Lecture-1a)

```
table(recidivism$Recidivism_Arrest_Year1)
```

```
    0
18111
```

```
table(recidivism$Recidivism_Arrest_Year2)
```

```
    0     1
13544  4567
```

```
table(recidivism$Recidivism_Arrest_Year3)
```

```
    0     1
15498  2613
```

```
table(recidivism$Training_Sample)
```

```
    0     1
 5460 12651
```

1.4.1. Initial data preparation

We will copy/paste the blueprint we developed earlier in the course (Lecture 2a)

```
# List of variable types

  outcome <- c('Recidivism_Arrest_Year2')

  categorical <- c('Residence_PUMA',
                   'Prison_Offense',
```

```
                  'Age_at_Release',
                  'Supervision_Level_First',
                  'Education_Level',
                  'Prison_Years',
                  'Gender',
                  'Race',
                  'Gang_Affiliated',
                  'Prior_Arrest_Episodes_DVCharges',
                  'Prior_Arrest_Episodes_GunCharges',
                  'Prior_Conviction_Episodes_Viol',
                  'Prior_Conviction_Episodes_PPViolationCharges',
                  'Prior_Conviction_Episodes_DomesticViolenceCharges',
                  'Prior_Conviction_Episodes_GunCharges',
                  'Prior_Revocations_Parole',
                  'Prior_Revocations_Probation',
                  'Condition_MH_SA',
                  'Condition_Cog_Ed',
                  'Condition_Other',
                  'Violations_ElectronicMonitoring',
                  'Violations_Instruction',
                  'Violations_FailToReport',
                  'Violations_MoveWithoutPermission',
                  'Employment_Exempt')

  numeric   <- c('Supervision_Risk_Score_First',
                 'Dependents',
                 'Prior_Arrest_Episodes_Felony',
                 'Prior_Arrest_Episodes_Misd',
                 'Prior_Arrest_Episodes_Violent',
                 'Prior_Arrest_Episodes_Property',
                 'Prior_Arrest_Episodes_Drug',
                 'Prior_Arrest_Episodes_PPViolationCharges',
                 'Prior_Conviction_Episodes_Felony',
                 'Prior_Conviction_Episodes_Misd',
                 'Prior_Conviction_Episodes_Prop',
                 'Prior_Conviction_Episodes_Drug',
                 'Delinquency_Reports',
                 'Program_Attendances',
                 'Program_UnexcusedAbsences',
                 'Residence_Changes',
                 'Avg_Days_per_DrugTest',
                 'Jobs_Per_Year')

  props      <- c('DrugTests_THC_Positive',
                  'DrugTests_Cocaine_Positive',
                  'DrugTests_Meth_Positive',
                  'DrugTests_Other_Positive',
                  'Percent_Days_Employed')

# 3) Convert all nominal, ordinal, and binary variables to factors
  # Leave the rest as is

  for(i in categorical){
```

```
    recidivism[,i] <- as.factor(recidivism[,i])

  }

# 4) For variables that represent proportions, add/substract a small number
  # to 0s/1s for logit transformation

  for(i in props){
    recidivism[,i] <- ifelse(recidivism[,i]==0,.0001,recidivism[,i])
    recidivism[,i] <- ifelse(recidivism[,i]==1,.9999,recidivism[,i])
  }


################################################################################

# Blueprint for processing variables

require(recipes)

blueprint <- recipe(x  = recidivism,
                    vars  = c(categorical,numeric,props,outcome),
                    roles = c(rep('predictor',48),'outcome')) %>%
  step_indicate_na(all_of(categorical),all_of(numeric),all_of(props)) %>%
  step_zv(all_numeric()) %>%
  step_impute_mean(all_of(numeric),all_of(props)) %>%
  step_impute_mode(all_of(categorical)) %>%
  step_logit(all_of(props)) %>%
  step_ns(all_of(numeric),all_of(props),deg_free=3) %>%
  step_normalize(paste0(numeric,'_ns_1'),
                 paste0(numeric,'_ns_2'),
                 paste0(numeric,'_ns_3'),
                 paste0(props,'_ns_1'),
                 paste0(props,'_ns_2'),
                 paste0(props,'_ns_3')) %>%
  step_dummy(all_of(categorical),one_hot=TRUE) %>%
  step_num2factor(Recidivism_Arrest_Year2,
                  transform = function(x) x + 1,
                  levels=c('No','Yes'))

  # The final step_num2factor is necessary to be able to work with the
  # caret::train() function
  # This syntax simply converts 0s and 1s to 1s and 2s, then convert it to a factor
  # with labels attached (No:1, Yes:2)

blueprint
```

```
Recipe

Inputs:

      role #variables
   outcome          1
 predictor         48
```

```
Operations:

Creating missing data variable indicators for all_of(categorical), all_of(numeric), all_of(pr...
Zero variance filter on all_numeric()
Mean Imputation for all_of(numeric), all_of(props)
Mode Imputation for all_of(categorical)
Logit transformation on all_of(props)
Natural Splines on all_of(numeric), all_of(props)
Centering and scaling for paste0(numeric, "_ns_1"), paste0(numeric, "_ns_...
Dummy variables from all_of(categorical)
Factor variables from Recidivism_Arrest_Year2
```

1.4.2. Train/Test split

We could do a random split of training and test sample. However, this dataset has already a filtering variable created by the agency who opened this competition. The test sample was unknown during the competition, and was later released after the competition was over. I will use the original training and testing datasets so we can compare the performance of our models to the highest ranking results in the competition.

```
loc <- which(recidivism$Training_Sample==1)

# Training dataset

recidivism_tr  <- recidivism[loc, ]
dim(recidivism_tr)
```

```
[1] 12651    54
```

```
# Test dataset

recidivism_te  <- recidivism[-loc, ]
dim(recidivism_te)
```

```
[1] 5460   54
```

1.4.3. Model fitting with the `caret` package

`caret` package has `glm` available as an engine to use.

```
require(caret)

# getModelInfo()$glm
```

For 10-fold cross-validation, I will first do a random shuffle of the data and then assign each individual to a fold. Note that I added two arguments, `summaryFunction = mnLogLoss` and `classProbs=TRUE` for the `trainControl()` function. This is necessary to fit the logistic regression by minimizing the negative logistic loss function (or maximizing the logistic loss).

NOTE

`mnLogLoss` yields computation of **minus log-likelihood**. The negative of loglikelihood is also sometimes called **the cross-entropy loss function**. Technically, maximizing the logistic loss function is equivalent to minimizing the cross-entropy loss function. When you use `glmnet` and specify `summaryFunction = mnLogLoss` in the cross-validation settings, it is minimizing the negative of the logistic loss function. You should be aware of it because it may be confusing or misleading when you start tuning parameters based on the value of logloss reported by the `glmnet` function. When you are using `caret::train()` for training a `glmnet` model with a binary outcome, keep in mind that a lower value o logloss function is better because the function actually minimizes the negative of the logloss function.

```
# Cross validation settings

  # Create the index values for 10-folds to provide to the
  # trainControl function. This way, you can reproduce the results in the future
  # and use the same folds across models.

    # Randomly shuffle the data

      set.seed(10302021) # for reproducibility

      recidivism_tr = recidivism_tr[sample(nrow(recidivism_tr)),]

    # Create 10 folds with equal size

      folds = cut(seq(1,nrow(recidivism_tr)),breaks=10,labels=FALSE)

    # Create the list for each fold

      my.indices <- vector('list',10)
      for(i in 1:10){
        my.indices[[i]] <- which(folds!=i)
      }


      cv <- trainControl(method    = "cv",
                   index           = my.indices,
                   classProbs      = TRUE,
                   summaryFunction = mnLogLoss)
```

Now, we can train the logistic regression model using 10-fold cross validation.

```
# Train the model

caret_mod <- caret::train(blueprint,
                          data      = recidivism_tr,
                          method    = "glm",
                          family    = 'binomial',
                          metric    = 'logLoss',
                          trControl = cv)

caret_mod
```

After we train the model, we can predict the probabilities for the observation in the test dataset.

```
# Predict the probabilities for the observations in the test dataset

predicted_te <- predict(caret_mod, recidivism_te, type='prob')

dim(predicted_te)
```

```
[1] 5460    2
```

```
head(predicted_te)
```

```
         No       Yes
1 0.8903231 0.1096769
2 0.7798314 0.2201686
3 0.7053433 0.2946567
4 0.6706977 0.3293023
5 0.7429843 0.2570157
6 0.8394731 0.1605269
```

Finally, we can calculate several evaluation metrics (e.g., AUC, TPR, TNR, FPR, Precision) for the model.

```
# Compute the AUC

require(cutpointr)

cut.obj <- cutpointr(x     = predicted_te$Yes,
                     class = recidivism_te$Recidivism_Arrest_Year2)

auc(cut.obj)
```

```
[1] 0.7191757
```

```
# Confusion matrix assuming the threshold is 0.5

pred_class <- ifelse(predicted_te$Yes>.5,1,0)

confusion <- table(recidivism_te$Recidivism_Arrest_Year2,pred_class)

confusion
```

```
   pred_class
       0    1
  0 3936  210
  1 1127  187
```

```
# True Negative Rate

confusion[1,1]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.9493488
```

```
# False Positive Rate

confusion[1,2]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.05065123
```

```
# True Positive Rate

confusion[2,2]/(confusion[2,1]+confusion[2,2])
```

```
[1] 0.1423135
```

```
# Precision

confusion[2,2]/(confusion[1,2]+confusion[2,2])
```

```
[1] 0.4710327
```

Regularization in Logistic Regression

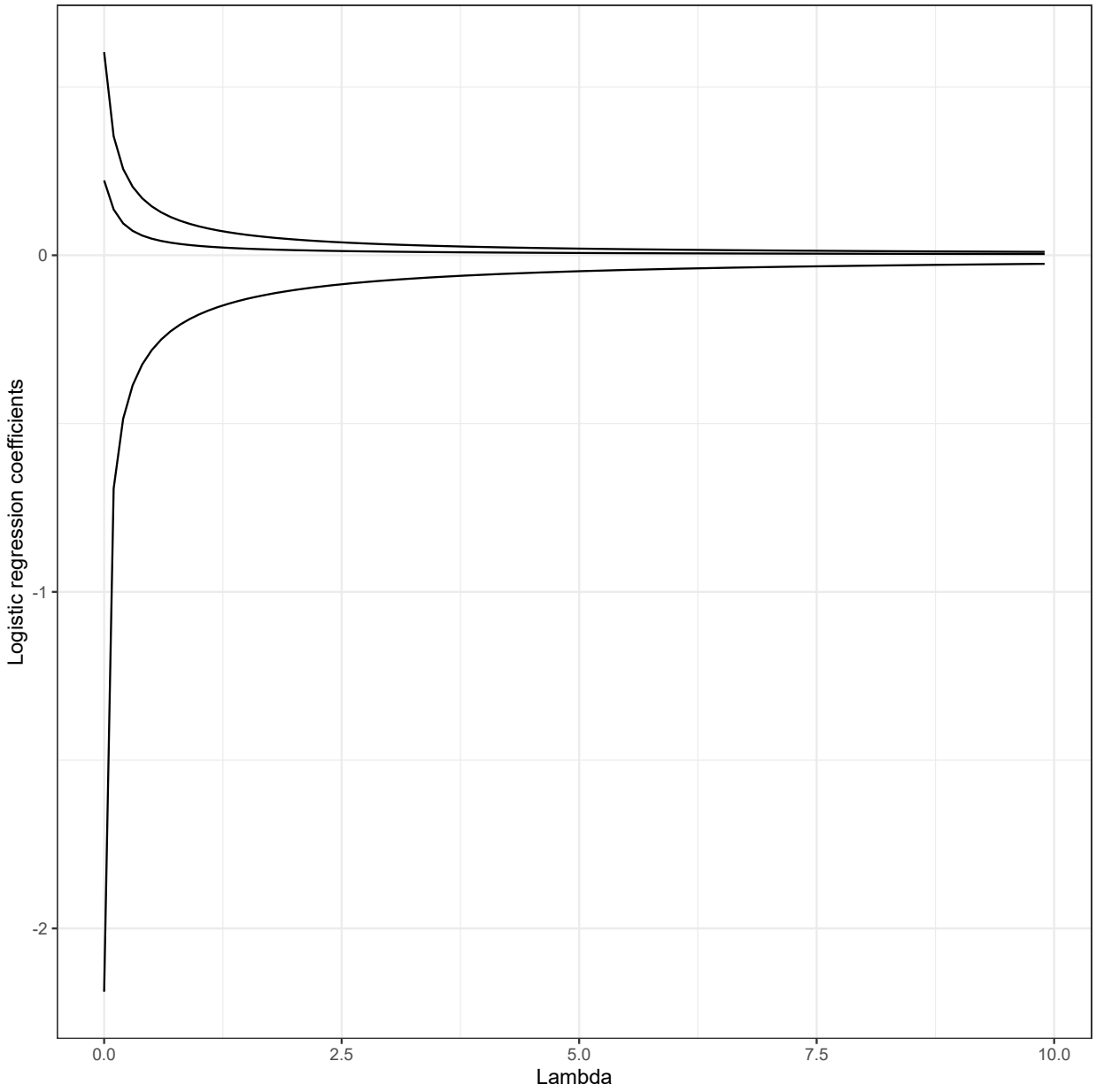
The regularization works very similarly in logistic regression as it was discussed in linear regression. We add penalty terms to the loss function to avoid large coefficients. By including a penalty term, we reduce model variance in exchange of adding bias. By optimizing the degree of penalty via tuning, we can typically get models with better performance than a logistic regression with no regularization.

Ridge Penalty

The loss function with ridge penalty applied in logistic regression is the following:

$$\log\mathfrak{L}(\mathbf{Y}|\boldsymbol{\beta}) = \left(\sum_{i=1}^{N} Y_i \times ln(P_i) + (1 - Y_i) \times ln(1 - P_i)\right) - \frac{\lambda}{2}\sum_{i=1}^{P} \beta_p^2$$

Notice that we now substract the penalty from the logistic loss, because we are maximizing the quantity. The penalty term has the same effect in the logistic regression, and it will pull the regression coefficients toward zero, but will not make them exactly equal to zero. Below, you can see a plot of change in logistic regression coefficients for some of the variables from a model that will be fit in the next section at increasing levels of ridge penalty.



NOTE

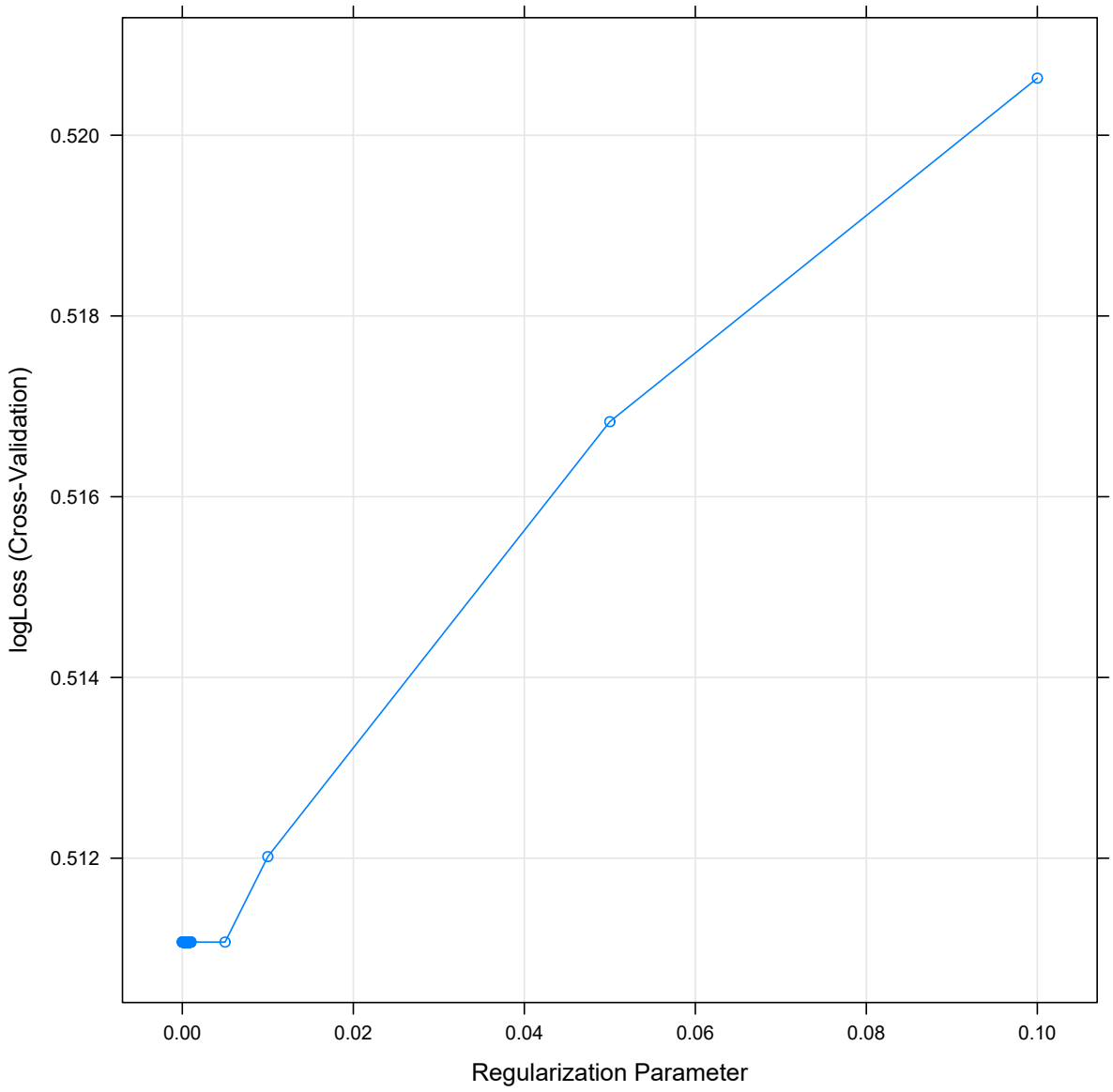
The `glmnet` package divides the value of loss function by sample size (N) during the optimization (Equation 12 and 2-3 in this paper). This technically does not change anything in terms of the optimal solution. On the other hand, we should be aware that the ridge penalty being applied in the 'glmnet package has become

$$N\frac{\lambda}{2}\sum_{i=1}^{P}\beta_p^2.$$

This information may be important while searching plausible values of λ for the `glmnet` package.

Model Fitting with the caret package

```
# Cross-validation settings

cv <- trainControl(method    = "cv",
                   index           = my.indices,
                   classProbs      = TRUE,
                   summaryFunction = mnLogLoss)

# Hyperparameter tuning grid for ridge penalty (lambda), alpha = 0

grid <- data.frame(alpha = 0, lambda = c(seq(0,.001,.00001),.005,.01,.05,.1))
grid

# Train the model

caret_logistic_ridge <- caret::train(blueprint,
                                     data      = recidivism_tr,
                                     method    = "glmnet",
                                     family    = 'binomial',
                                     metric    = 'logLoss',
                                     trControl = cv,
                                     tuneGrid  = grid)

caret_logistic_ridge

# check the results

plot(caret_logistic_ridge)

caret_logistic_ridge$bestTune
```




```
    alpha lambda
102     0  0.005
```

```
# Predict the probabilities for the observations in the test dataset

predicted_te <- predict(caret_logistic_ridge, recidivism_te, type='prob')

dim(predicted_te)
```

```
[1] 5460    2
```

```
head(predicted_te)
```

```
         No       Yes
1 0.8679515 0.1320485
2 0.7793438 0.2206562
3 0.7242196 0.2757804
4 0.7020330 0.2979670
5 0.7313716 0.2686284
6 0.8399176 0.1600824
```

```
# Compute the AUC

require(cutpointr)

cut.obj <- cutpointr(x     = predicted_te$Yes,
                     class = recidivism_te$Recidivism_Arrest_Year2)

auc(cut.obj)
```

```
[1] 0.7181268
```

```
# Confusion matrix assuming the threshold is 0.5

pred_class <- ifelse(predicted_te$Yes>.5,1,0)

confusion <- table(recidivism_te$Recidivism_Arrest_Year2,pred_class)

confusion
```

```
   pred_class
       0    1
  0 3957  189
  1 1152  162
```

```
# True Negative Rate

confusion[1,1]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.9544139
```

```
# False Positive Rate

confusion[1,2]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.04558611
```

```
# True Positive Rate

confusion[2,2]/(confusion[2,1]+confusion[2,2])
```

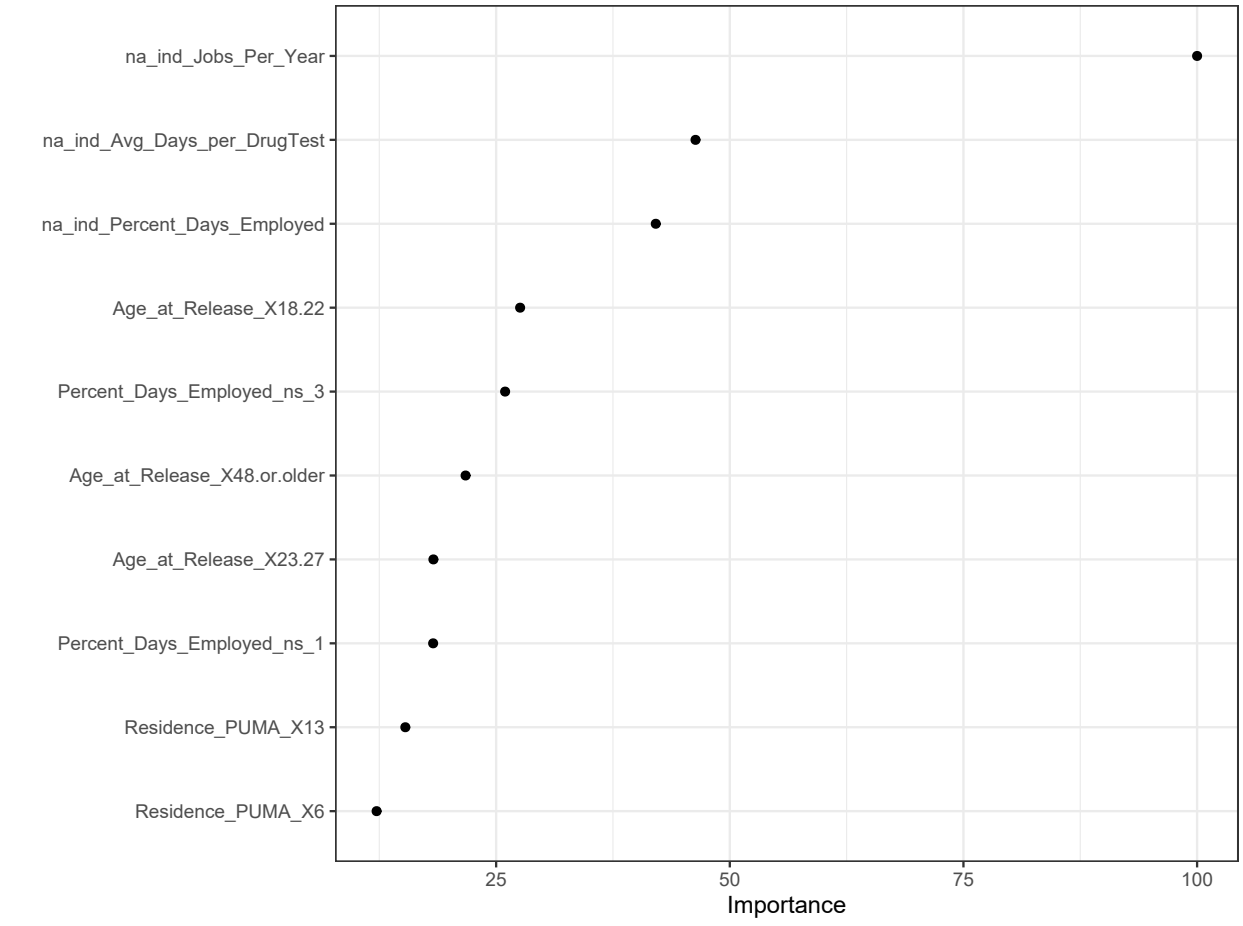
```
[1] 0.1232877
```

```
# Precision

confusion[2,2]/(confusion[1,2]+confusion[2,2])
```

```
[1] 0.4615385
```

Variable Importance

```
#install.packages('vip')

require(vip)

vip(caret_logistic_ridge,
    num_features = 10,
    geom = "point") +
  theme_bw()
```




```
coefs <- coef(caret_logistic_ridge$finalModel,caret_logistic_ridge$bestTune$lambda)

ind   <- order(abs(coefs[,1]),decreasing=T)
```

```
head(as.matrix(coefs[ind,]),10)
```

```
                                [,1]
na_ind_Jobs_Per_Year         -2.1879897
na_ind_Avg_Days_per_DrugTest -1.0139222
na_ind_Percent_Days_Employed -0.9208957
(Intercept)                  -0.8620865
Age_at_Release_X18.22         0.6033616
Percent_Days_Employed_ns_3   -0.5682384
Age_at_Release_X48.or.older  -0.4756952
Age_at_Release_X23.27         0.4003676
Percent_Days_Employed_ns_1   -0.3997522
Residence_PUMA_X13            0.3346626
```

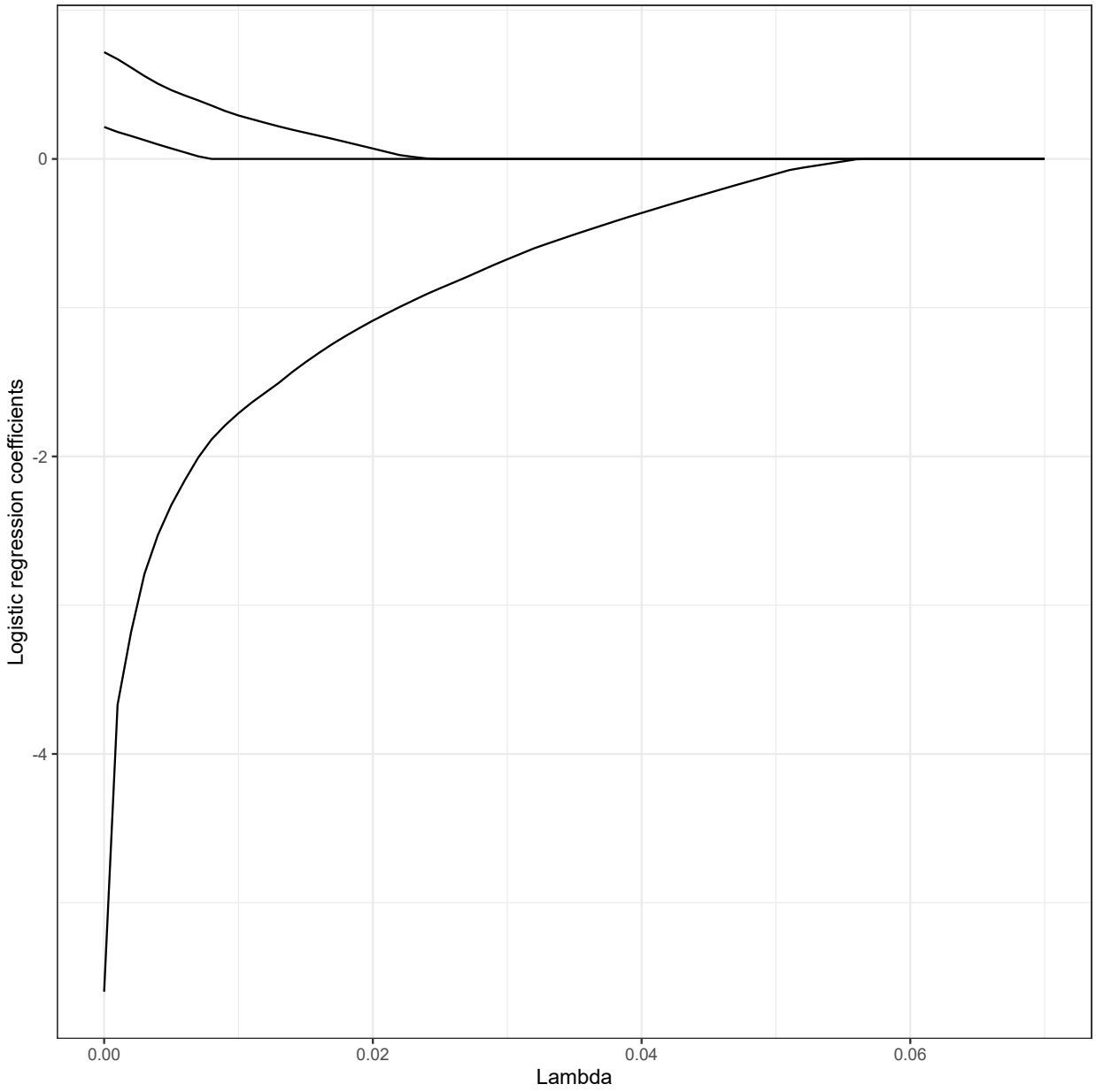
Lasso Penalty

The loss function with the lasso penalty applied in logistic regression is the following:

$$\log\mathcal{L}(\mathbf{Y}|\boldsymbol{\beta}) = \left(\sum_{i=1}^{N} Y_i \times ln(P_i) + (1 - Y_i) \times ln(1 - P_i)\right) - \lambda \sum_{i=1}^{P} |\beta_p|$$

The penalty terms has the same effect in the logistic regression, and it will make the regression coefficients equal to zero when a sufficiently large λ value is applied.

Below, you can see a plot of change in logistic regression coefficients for some of the variables from a model that will be fit in the next section at increasing levels of lasso penalty.



NOTE

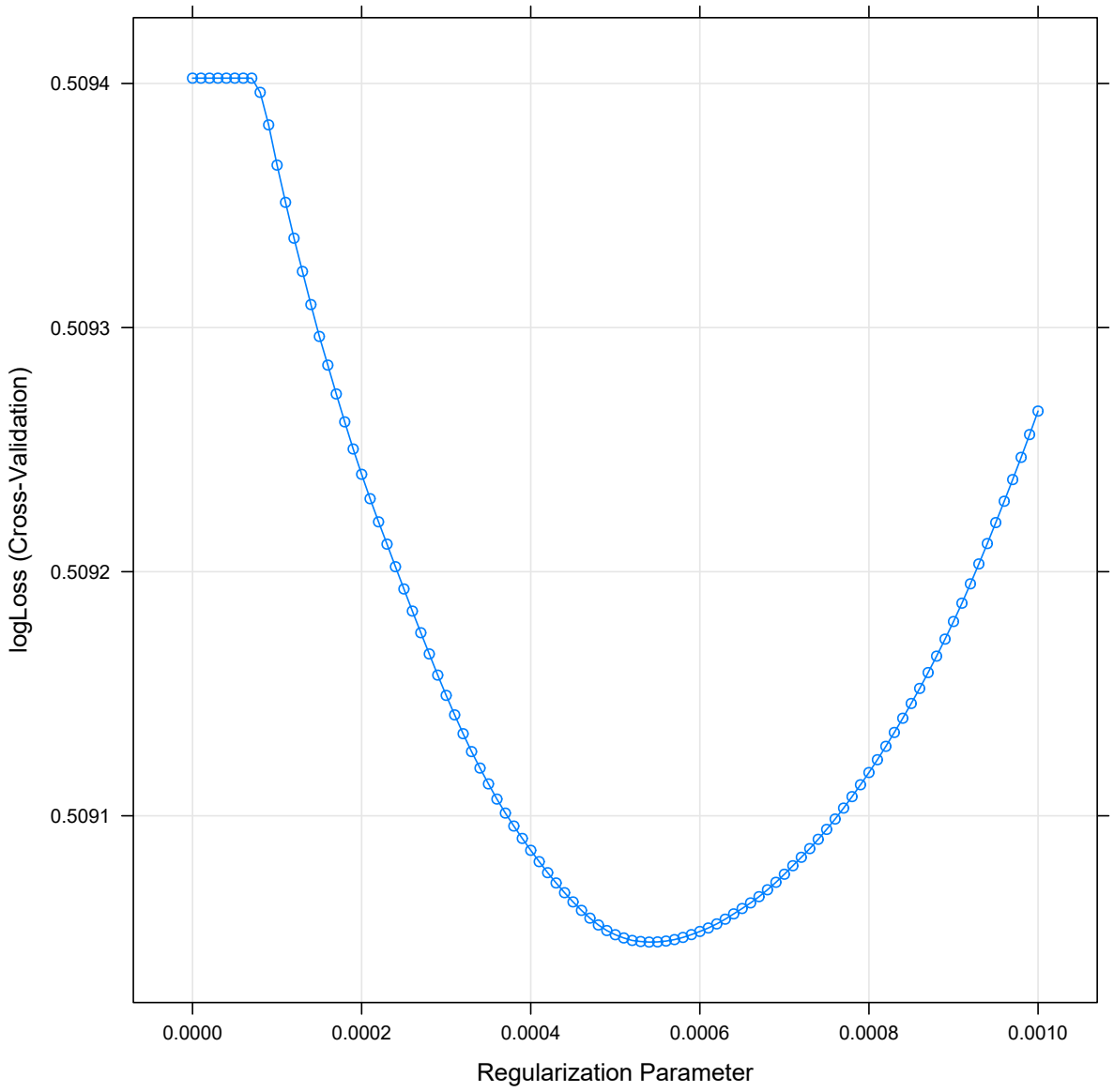
Note that `glmnet` package divides the value of loss function by sample sample (N) during the optimization (Equation 12 and 2-3 in this paper). You should be aware that the lasso penalty being applied in the 'glmnet package has become

$$N\lambda \sum_{i=1}^{P} |\beta_p|.$$

This information may be important while searching plausible values of λ for the `glmnet` package.

Model Fitting with the caret package

```
# Cross-validation settings

cv <- trainControl(method     = "cv",
                   index           = my.indices,
                   classProbs      = TRUE,
                   summaryFunction = mnLogLoss)

# Hyperparameter tuning grid for ridge penalty (lambda), alpha = 0

grid <- data.frame(alpha = 1, lambda = seq(0,.001,.00001))
grid

  # If you want to fit logistic regression you can also set lambda to 0
  # grid <- data.frame(alpha = 0, lambda = 0)
  # This is equivalent to glm fit we did in the previous section

# Train the model

caret_logistic_lasso <- caret::train(blueprint,
                                     data      = recidivism_tr,
                                     method    = "glmnet",
                                     family    = 'binomial',
                                     metric    = 'logLoss',
                                     trControl = cv,
                                     tuneGrid  = grid)

caret_logistic_lasso

# check the results

plot(caret_logistic_lasso)

caret_logistic_lasso$bestTune
```




```
   alpha  lambda
55     1 0.00054
```

```
# Predict the probabilities for the observations in the test dataset

predicted_te <- predict(caret_logistic_lasso, recidivism_te, type='prob')

dim(predicted_te)
```

```
[1] 5460    2
```

```
head(predicted_te)
```

```
         No       Yes
1 0.8814907 0.1185093
2 0.7668175 0.2331825
3 0.7145579 0.2854421
4 0.6785981 0.3214019
5 0.7426930 0.2573070
6 0.8344650 0.1655350
```

```
# Compute the AUC

require(cutpointr)

cut.obj <- cutpointr(x     = predicted_te$Yes,
                     class = recidivism_te$Recidivism_Arrest_Year2)

auc(cut.obj)
```

```
[1] 0.7199856
```

```
# Confusion matrix assuming the threshold is 0.5

pred_class <- ifelse(predicted_te$Yes>.5,1,0)

confusion <- table(recidivism_te$Recidivism_Arrest_Year2,pred_class)

confusion
```

```
   pred_class
       0    1
  0 3948  198
  1 1147  167
```

```
# True Negative Rate

confusion[1,1]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.9522431
```

```
# False Positive Rate

confusion[1,2]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.04775687
```

```
# True Positive Rate

confusion[2,2]/(confusion[2,1]+confusion[2,2])
```

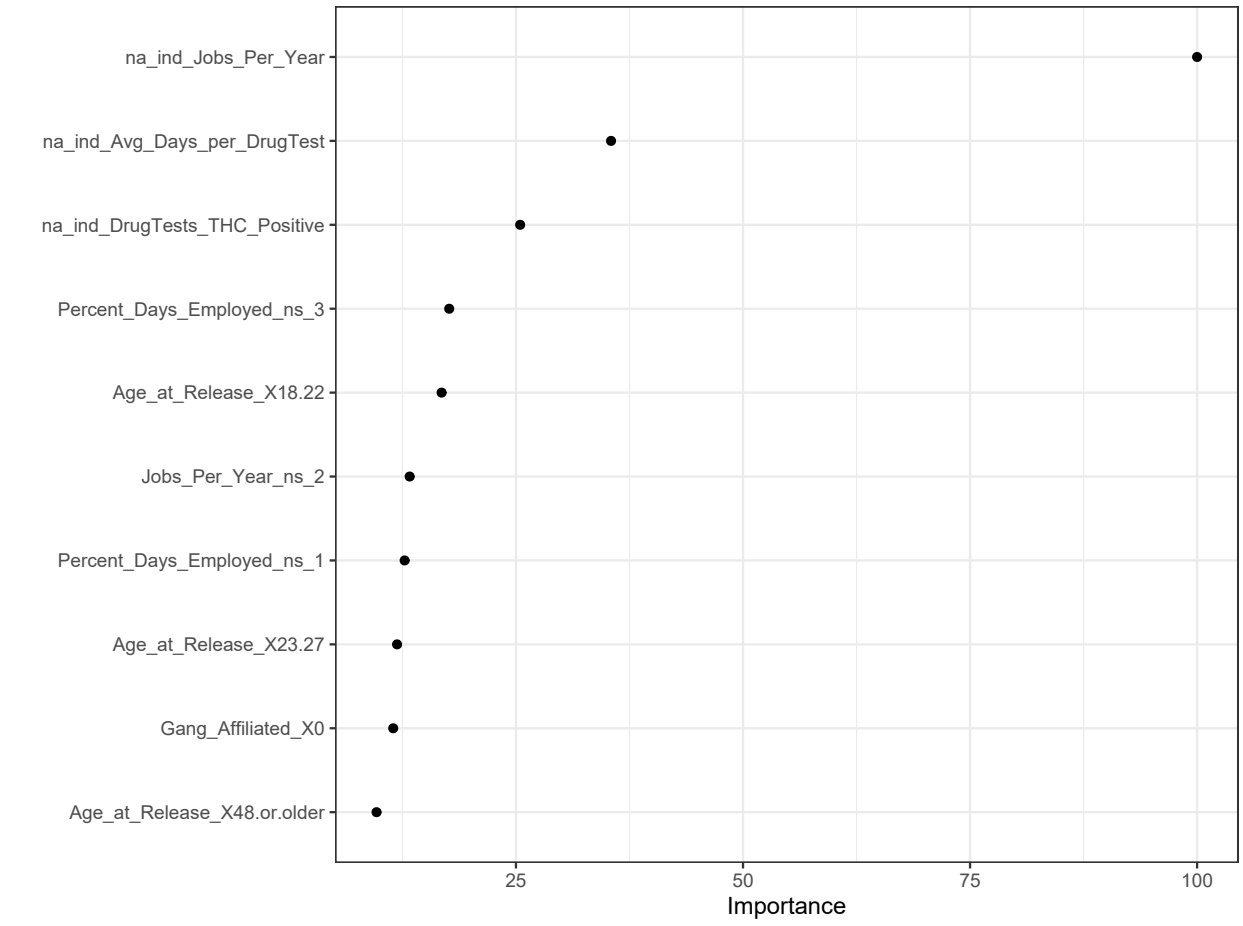
```
[1] 0.1270928
```

```
# Precision

confusion[2,2]/(confusion[1,2]+confusion[2,2])
```

```
[1] 0.4575342
```

Variable Importance

```
#install.packages('vip')

require(vip)

vip(caret_logistic_lasso,
    num_features = 10,
    geom = "point") +
  theme_bw()
```




```
coefs <- coef(caret_logistic_lasso$finalModel,
              caret_logistic_lasso$bestTune$lambda)
```

```
ind   <- order(abs(coefs[,1]),decreasing=T)

head(as.matrix(coefs[ind,]),10)
```

```
                                  [,1]
na_ind_Jobs_Per_Year          -4.1414442
na_ind_Avg_Days_per_DrugTest  -1.4689557
na_ind_DrugTests_THC_Positive  1.0544674
Percent_Days_Employed_ns_3    -0.7309134
Age_at_Release_X18.22          0.6962768
Jobs_Per_Year_ns_2             0.5506316
Percent_Days_Employed_ns_1    -0.5277132
Age_at_Release_X23.27          0.4931343
Gang_Affiliated_X0            -0.4756509
Age_at_Release_X48.or.older   -0.3996344
```

Elastic Net

The loss function with the elastic applied is the following:

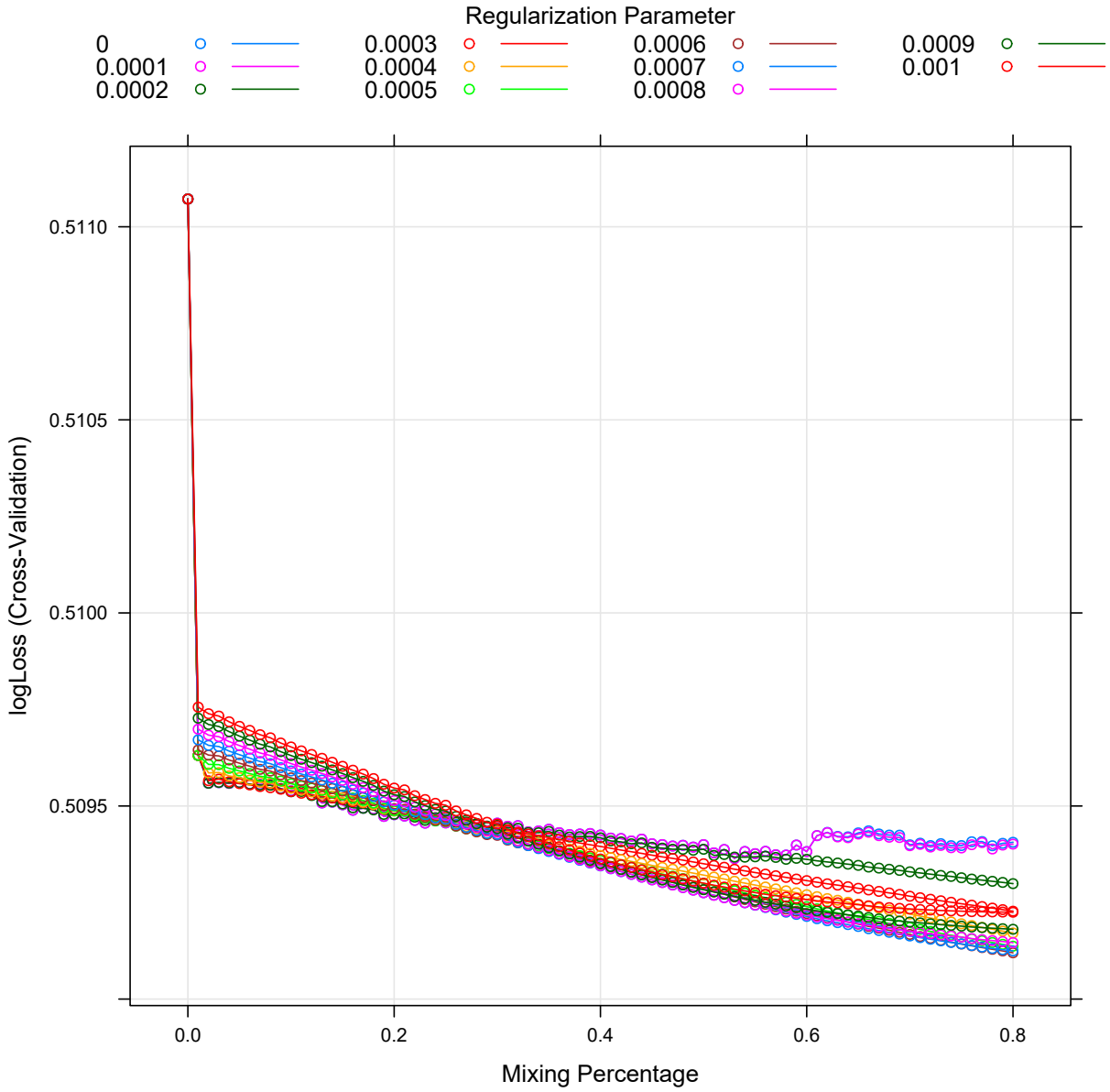
$$\log\mathfrak{L}(\mathbf{Y}|\beta) = \left(\sum_{i=1}^{N} Y_i \times ln(P_i) + (1 - Y_i) \times ln(1 - P_i)\right) - \left((1-\alpha)\frac{\lambda}{2}\sum_{i=1}^{P}\beta_p^2 + \alpha\lambda\sum_{i=1}^{P}|\beta_p|\right)$$

Model Fitting with the caret package

```
# Cross-validation settings

cv <- trainControl(method     = "cv",
                   index           = my.indices,
                   classProbs      = TRUE,
                   summaryFunction = mnLogLoss)

# Hyperparameter tuning grid for ridge penalty (lambda), alpha = 0

  grid <- expand.grid(alpha = seq(0,.8,.01), lambda = seq(0,.001,.0001))
  grid

# Train the model

caret_logistic_elastic <- caret::train(blueprint,
                                       data      = recidivism_tr,
                                       method    = "glmnet",
                                       family    = 'binomial',
                                       metric    = 'logLoss',
                                       trControl = cv,
                                       tuneGrid  = grid)

caret_logistic_elastic

# check the results
```

```
plot(caret_logistic_elastic)

caret_logistic_elastic$bestTune
```

```
0.5091198
```




```
    alpha lambda
887   0.8 0.0006
```

```
# Predict the probabilities for the observations in the test dataset
```

```
predicted_te <- predict(caret_logistic_elastic, recidivism_te, type='prob')

dim(predicted_te)
```

```
[1] 5460    2
```

```
head(predicted_te)
```

```
         No       Yes
1 0.8813683 0.1186317
2 0.7682286 0.2317714
3 0.7144253 0.2855747
4 0.6792849 0.3207151
5 0.7425723 0.2574277
6 0.8351791 0.1648209
```

```
# Compute the AUC

require(cutpointr)

cut.obj <- cutpointr(x     = predicted_te$Yes,
                     class = recidivism_te$Recidivism_Arrest_Year2)

auc(cut.obj)
```

```
[1] 0.7199566
```

```
# Confusion matrix assuming the threshold is 0.5

pred_class <- ifelse(predicted_te$Yes>.5,1,0)

confusion <- table(recidivism_te$Recidivism_Arrest_Year2,pred_class)

confusion
```

```
   pred_class
       0    1
  0 3947  199
  1 1147  167
```

```
# True Negative Rate

confusion[1,1]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.9520019
```

```
# False Positive Rate

confusion[1,2]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.04799807
```

```
# True Positive Rate

confusion[2,2]/(confusion[2,1]+confusion[2,2])
```

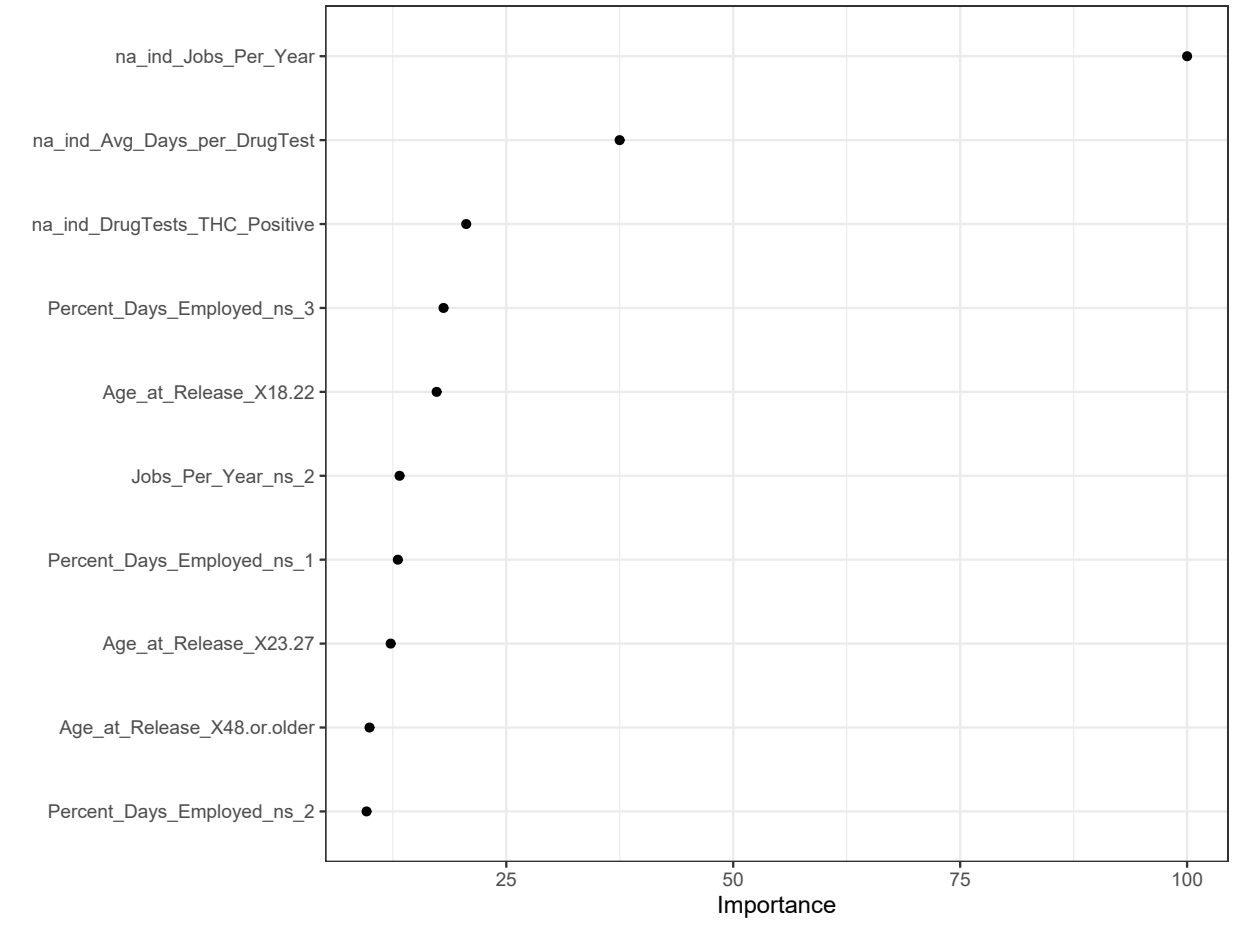
```
[1] 0.1270928
```

```
# Precision

confusion[2,2]/(confusion[1,2]+confusion[2,2])
```

```
[1] 0.4562842
```

Variable Importance

```
#install.packages('vip')

require(vip)

vip(caret_logistic_elastic,
    num_features = 10,
    geom = "point") +
  theme_bw()
```



```
coefs <- coef(caret_logistic_elastic$finalModel,
              caret_logistic_elastic$bestTune$lambda)

ind   <- order(abs(coefs[,1]),decreasing=T)

head(as.matrix(coefs[ind,]),10)
```

```
                                  [,1]
na_ind_Jobs_Per_Year          -4.0314714
na_ind_Avg_Days_per_DrugTest  -1.5111465
(Intercept)                   -0.8870970
na_ind_DrugTests_THC_Positive  0.8298998
Percent_Days_Employed_ns_3    -0.7291377
Age_at_Release_X18.22          0.6984920
Jobs_Per_Year_ns_2             0.5338077
Percent_Days_Employed_ns_1    -0.5262051
Age_at_Release_X23.27          0.4944036
Age_at_Release_X48.or.older   -0.4003761
```

Comparison of Models

Performance evaluation metrics on the test data.

	-LL	AUC	ACC	TPR	TNR	FPR	PRE
Logistic Regression	0.5096	0.7192	0.755	0.142	0.949	0.051	0.471
Logistic Regression with Ridge Penalty	0.5111	0.7181	0.754	0.123	0.954	0.046	0.461
Logistic Regression with Lasso Penalty	0.5090	0.7200	0.754	0.127	0.952	0.048	0.458
Logistic Regression with Elastic Net	0.5091	0.7200	0.753	0.127	0.952	0.048	0.456