Implementation of LASSO-Logistic regression.

Hung Pham

4/21/2020

1 Synopsis

This project showcases the manual implementation of the LASSO algorithm with logistic regression in R. We compare our implementation to the gold-standard implementation in the R package `{glmnet}` by Jerome Friedman et. al. in classifying malignant (M) and benign (B) tumors from images of breast cancer tumors recorded in the `breast-cancer.csv` data of from the UCI machine learning repository.

We show our manual implementations in R of the following algorithms:
- Newton-Raphson optimizer with step-halving and re-direction steps
- Logistic regression algorithm, using Newton-Raphson optimizer
- LASSO regression extension to logistic regression, using a coordinate descent optimizer

Our implementation are shown to be comparable to gold-standard implementations in R. The source code can be found in the source_code folder.

2 Data description

The data `breast-cancer.csv` have 569 row and 33 columns. From the UCI repository:

> Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image

There are 357 benign and 212 malignant cases in the `diagnosis` column which will be our outcome variable for logistic regression. There are 30 columns corresponding to the mean, standard deviation, and largest values (points on the tails) of the following 10 features computed for each cell nucleus.

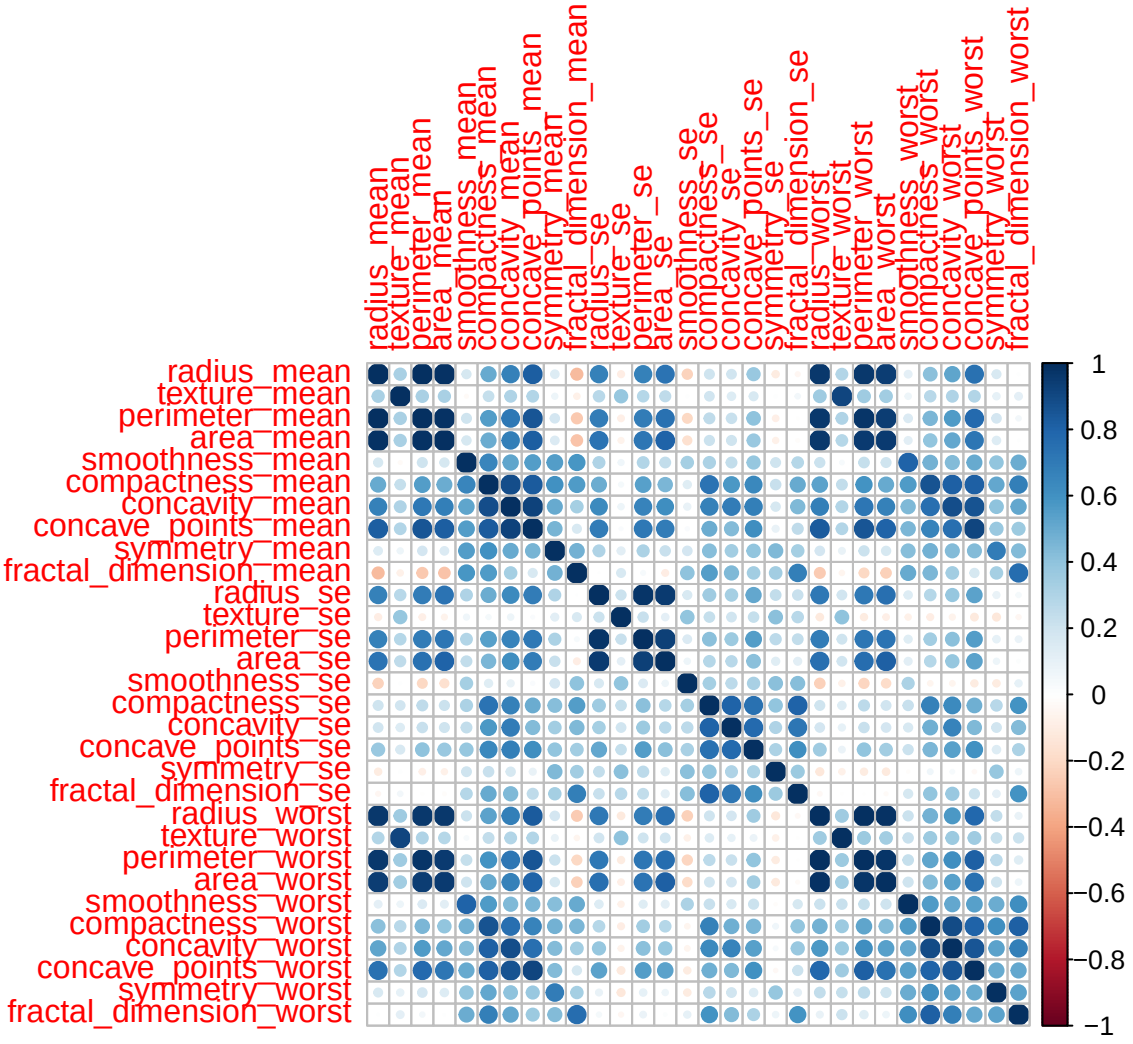
- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^ 2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension (“coastline approximation” - 1)

Our goal is to use LASSO-logistic regression to build a predictive model using the 30 feature columns to predict if a case is either malignant or benign.

2.1 Data import and clean-up

We import the data and study the correlation structure.

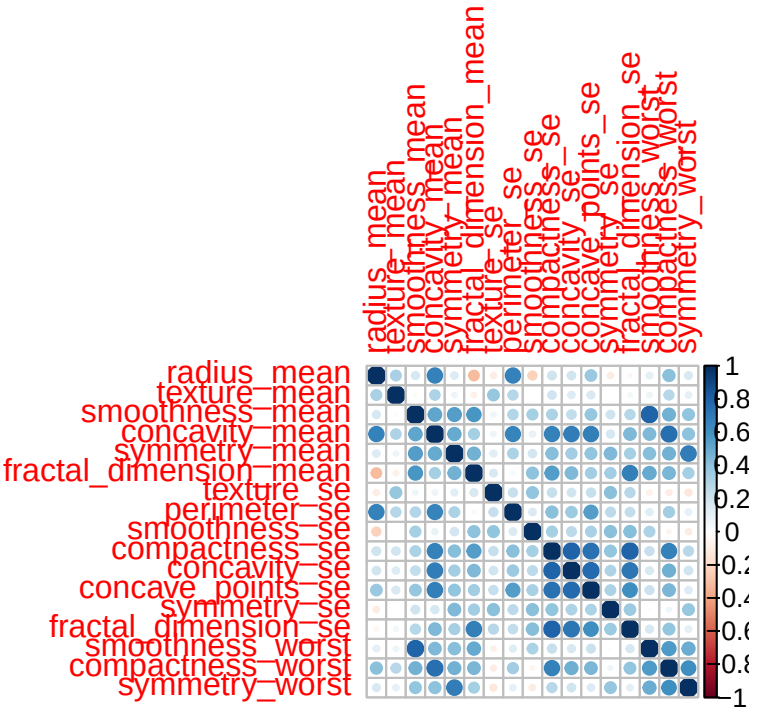
```
breast_raw<- read.csv(paste0(
        here::here(),"/data/breast-cancer.csv"),
                  row.names = NULL) %>% janitor::clean_names() %>%
        select(-id,-x)

corr_matrix <- cor(breast_raw %>% select(-diagnosis,))
corrplot(corr_matrix)
```



We scale and center our data. Also, we remove the variables that have correlation > 0.9. In the end, we have 20 variables.

```
breast<-breast_raw %>% select(-perimeter_mean,-area_mean,-radius_worst,
                              -radius_se,-perimeter_worst,-area_worst,
                              -texture_worst,-concave_points_worst,
                              -area_se,-concave_points_mean,
                              -compactness_mean,-concavity_worst,
```

```
                              -fractal_dimension_worst
                              ) %>%
        mutate(across(.cols = -diagnosis,
                      .fns = function(x){return((x-mean(x))/sd(x))}))

breast_no_int<-breast %>% mutate(diagnosis = ifelse(diagnosis == "M",1,0))
breast_int<- cbind(intercept = 1,breast_no_int)

corr_matrix_1 <- cor(breast_no_int %>% select(-diagnosis,))
corrplot(corr_matrix_1)
```



This correlogram correspond to the cleaned version of the data we are using to fit our results.

3 Implementations

3.1 Logistic regression

A logistic regression algorithm is used to regress a linear combination of features to a continuous outcome (probability from 0 to 1) using the logit link function. To accomplish this, we need two parts: 1) a function that calculate the likelihood, gradient, and hessian of our model, and 2) an optimizer that finds the maximum likelihood using the gradient and hessian as the input. The following is the former part. The likelihood is calculated as:

$$\sum \mathbf{Y} \cdot (\mathbf{X}\beta - log(1 + exp(\mathbf{X}\beta)) + (1 - \mathbf{Y}) \cdot (-log(1 + exp(\mathbf{X}\beta)))$$

The gradient is calculated as:

$$\mathbf{X}(\mathbf{Y} - \frac{exp(\mathbf{X}\beta)}{1 + exp(\mathbf{X}\beta)})$$

The Hessian is calculated as:

$$-\mathbf{X}^T diag(\frac{exp(\mathbf{X}\beta)}{1+exp(\mathbf{X}\beta)})\mathbf{X}$$

```
logistic_mle <- function(dat, beta_vec, response = "diagnosis", refit = T) {
        # y_mat is the outcome column
        y_mat <- matrix(as.vector(dat[[response]]), ncol = 1)

        # x_mat is the feature dataset, with an extra column of 1 for intercept
        x_mat <- cbind(1,as.matrix(dat[, which(names(dat) != response)]))

        mu <- x_mat %*% beta_vec
        pi <- exp(x_mat %*% beta_vec)

        likelihood <- sum(y_mat * (mu - log(1 + pi))
                          + (1 - y_mat) * (-log(1 + pi)))

        grad <- t(x_mat) %*% (y_mat - pi / (1 + pi))

        hess <- -t(x_mat) %*% diag(as.vector(pi / (1 + pi) ^ 2)) %*% x_mat

        return(list(loglik = likelihood, grad = as.vector(grad), Hess = hess))
}
```

3.2 Newton-Raphson optimizer

The Newton-Raphson optimizer, with step-halving and re-direction steps, is used to find the direction of the steepest descent to a local minima (or equivalently ascent to a local maxima with the opposite sign) by solving for the system of equations of gradient = 0. More details can be found in this wikipedia page. We use this to find the MLE of our logistic regression algorithm.

```
newton_raphson <- function(dat, fun, start, tol=1e-10,
                           maxiter = 200, n_halves = 10,
                           original_lambda = 1, beta_name = NULL) {
        i <- 0
        current <- start
        fit <- fun(dat, current)

        lambda <- original_lambda # To allow for halves stepping
        times_halved <- 0

        res <- c(0, fit$loglik, current, lambda, times_halved)

        prevloglik <- -Inf # To make sure it iterates

        while (i < maxiter && abs(fit$loglik - prevloglik) > tol)
        {
                i <- i + 1
                prevloglik <- fit$loglik
                prev <- current
```

```
                current <- prev - lambda * solve(fit$Hess) %*% fit$grad
                while (((fun(dat, current)$loglik < fun(dat, prev)$loglik) &&
                        times_halved < n_halves) |
                       is.infinite(fun(dat, current)$loglik)) {
                        times_halved <- times_halved + 1
                        lambda_1 <- lambda * (1 / 2) ^ (times_halved)
                        current <-
                                prev - lambda_1 * solve(fit$Hess) %*% fit$grad
                }

                fit <- fun(dat, current)
                res <- rbind(res, c(i, fit$loglik, current, lambda, times_halved))
                lambda <- original_lambda
                times_halved <- 0
        }
        colnames(res) <- c("iter","llik",
                           "intercept",beta_name,
                           "lambda", "times_halved")
        rownames(res) <- NULL
        res<-data.frame(round(res, 4))
        res$real_lambda = res$lambda * 0.5^(res$times_halved)
        estimates = tibble(term = c("intercept",beta_name),
                           coefficient = as.vector(current))

        return(list(iter = res,
                    estimates = estimates))
}
```

3.3 LASSO-Logistic Regression

The lasso logistic regression algorithm can use soft-thresholding and coordinate descent to find the solutions of the β estimates. With the implemented "warm start", our algorithm have a comparable rate of convergence to the Newton-Raphson algorithm. Details can be found in Hastie et. al. 2015.

```
soft_threshold = function(beta, lambda) {
        ifelse(abs(beta)>lambda && beta > 0,
               beta-lambda,
               ifelse(abs(beta) > lambda && beta < 0,
                      beta + lambda,0))
        }

logistic_lasso <- function(data, y, lambda, beta_vec, tol = 1e-10,
                           maxiter = 200, beta_name = NULL){
        i <- 0
        x_mat <- as.matrix(data)
        x_center<-attributes(x_mat)$`scaled:center`
        x_scale<-attributes(x_mat)$`scaled:scale`

        loglik <- 0
        res <- c(0, loglik, beta_vec)
        prevloglik <- Inf
```

```
        while (i < maxiter && abs(loglik - prevloglik) > tol && loglik < Inf){
                i <- i + 1
                prevloglik <- loglik
                for (k in 1:length(beta_vec)){
                        u <- x_mat %*% beta_vec
                        expu <- exp(u)
                        p <- expu/(1 + expu)
                        weight <- p*(1-p)

                        #Make sure weight doesn't overflow
                        weight <- ifelse(abs(0-weight) < 1e-7, 1e-7, weight)

                        #calculate working responses
                        zi <- u + (y-p)/weight

                        #calculate the residual
                        residual <- x_mat[,-k] %*% beta_vec[-k]

                        #calculate beta soft-threshold
                        beta_vec[k] <- soft_threshold(mean(weight*x_mat[,k]*
                         (zi-residual)),lambda)/(mean(weight*(x_mat[,k]^2)))}

                #Current Likelihood
                loglik <- 1/(2*nrow(x_mat))*sum(weight*(zi-x_mat%*%beta_vec)^2)
                + lambda*sum(abs(beta_vec))
                res <- rbind(res, c(i, loglik, beta_vec))
        }
        colnames(res)<-c("iter","llike",beta_name)
        estimates = tibble(term = beta_name,
                           coefficient = (as.vector(beta_vec)))

        return(list(iter = res,
                    estimates = estimates))
}
```

4 Results

4.1 Classical Logistic Regression with Newton-Raphson optimizer

We fit our logistic implementation to classify image "M" as 1 and "B" as 0. We use our Newton-Raphson function for optimization. We also compare our function estimate to {glmnet} estimates.

```
ans <- newton_raphson(breast_no_int,
                      logistic_mle,
                      rep(0,ncol(breast_no_int)),
                      maxiter = 200, n_halves = 20, original_lambda = 1,
                      beta_name = names(breast_no_int)
                      [which(names(breast_no_int) != "diagnosis")])

############################## FIT WITH GLMNET FOR COMPARISION ##################
test_dat <- breast_no_int %>%
        mutate(diagnosis  = factor(diagnosis , levels = c(0,1)))
```

```
my_rec<-recipe(diagnosis  ~., data = test_dat)

# With penalty term = 0, this is equivalent to a logistic regression model
my_model <- logistic_reg(penalty = 0, mixture = 1) %>%
        set_engine("glmnet") %>%
        set_mode("classification")

wf <- workflow() %>%
        add_recipe(my_rec) %>% # Workflow
        add_model(my_model)

# Compare the betas of the model with the true beta
fit_data <- wf %>%
        fit(test_dat) %>%
        pull_workflow_fit() %>% tidy()

################################################################################
logistic<-tibble(ans$estimates,
              glmnet_estimates = fit_data$estimate)%>%
        mutate(across(.cols = where(is.numeric),.fns = function(x)round(x,4))) %>%
        rename(newton_logistic = coefficient)
```

The following table shows the beta estimates of our logistic implementation and {glmnet}. Our function performs comparably to {glmnet}.

```
logistic %>% knitr::kable() %>%
        kable_classic_2(latex_options = "HOLD_position", position = "center")
```

term	newton_logistic	glmnet_estimates
intercept	-0.6229	-0.6217
radius_mean	3.7531	3.7014
texture_mean	1.7115	1.6854
smoothness_mean	-0.0290	0.0000
concavity_mean	3.9638	3.8181
symmetry_mean	-0.7934	-0.7028
fractal_dimension_mean	-0.9443	-0.8993
texture_se	0.3177	0.2808
perimeter_se	3.4739	3.3334
smoothness_se	-0.1801	-0.1550
compactness_se	-1.1806	-1.1236
concavity_se	-0.2796	-0.2342
concave_points_se	0.8336	0.7343
symmetry_se	-1.4438	-1.4010
fractal_dimension_se	-0.7971	-0.7674
smoothness_worst	1.7811	1.7219
compactness_worst	0.3102	0.2661
symmetry_worst	2.6700	2.5554

4.2 LASSO-logistic regression

4.2.1 Lambda = 0

We compare the results of our `logistic_lasso` function at $\lambda = 0$ to our Newton-Raphson logistic function and `{glmnet}` to make sure that soft-thresholding with $\lambda = 0$ yields the same results.

```
# separate x and y to calculate lambda max
breast_int_x<-breast_int[,which(names(breast_int) != "diagnosis")]
breast_int_y<-breast_int[,"diagnosis"]
# calculate lambda max

log_lasso<-logistic_lasso(data=breast_int_x,
                      y = breast_int_y,
                      lambda = 0,
                      beta_vec = as.matrix(rep(0,ncol(breast_int_x)),ncol = 1),
                      beta_name = c(names(breast_int_x)
                                    [which(names(breast_int_x) != "diagnosis")]))
log_lasso_res<-tibble(logistic,
                our_logistic_lasso = log_lasso$estimates$coefficient) %>%
        mutate(across(.cols = where(is.numeric),.fns = function(x)round(x,4)))
log_lasso_res %>% knitr::kable() %>%
        kable_classic_2(latex_options = "HOLD_position", position = "center")
```

term	newton_logistic	glmnet_estimates	our_logistic_lasso
intercept	-0.6229	-0.6217	-0.6227
radius_mean	3.7531	3.7014	3.7530
texture_mean	1.7115	1.6854	1.7114
smoothness_mean	-0.0290	0.0000	-0.0282
concavity_mean	3.9638	3.8181	3.9638
symmetry_mean	-0.7934	-0.7028	-0.7932
fractal_dimension_mean	-0.9443	-0.8993	-0.9448
texture_se	0.3177	0.2808	0.3178
perimeter_se	3.4739	3.3334	3.4743
smoothness_se	-0.1801	-0.1550	-0.1792
compactness_se	-1.1806	-1.1236	-1.1824
concavity_se	-0.2796	-0.2342	-0.2788
concave_points_se	0.8336	0.7343	0.8328
symmetry_se	-1.4438	-1.4010	-1.4438
fractal_dimension_se	-0.7971	-0.7674	-0.7970
smoothness_worst	1.7811	1.7219	1.7798
compactness_worst	0.3102	0.2661	0.3123
symmetry_worst	2.6700	2.5554	2.6699

Our estimates are very similar, indicating that our implementations works well.

4.2.2 Lambda > 0

We find the maximum λ that will shrink all our beta estimates to 0.

```
lambda_max<-max((t(as.matrix(breast_int_x)) %*% breast_int_y)/nrow(breast_int_x))
lambda_max
```

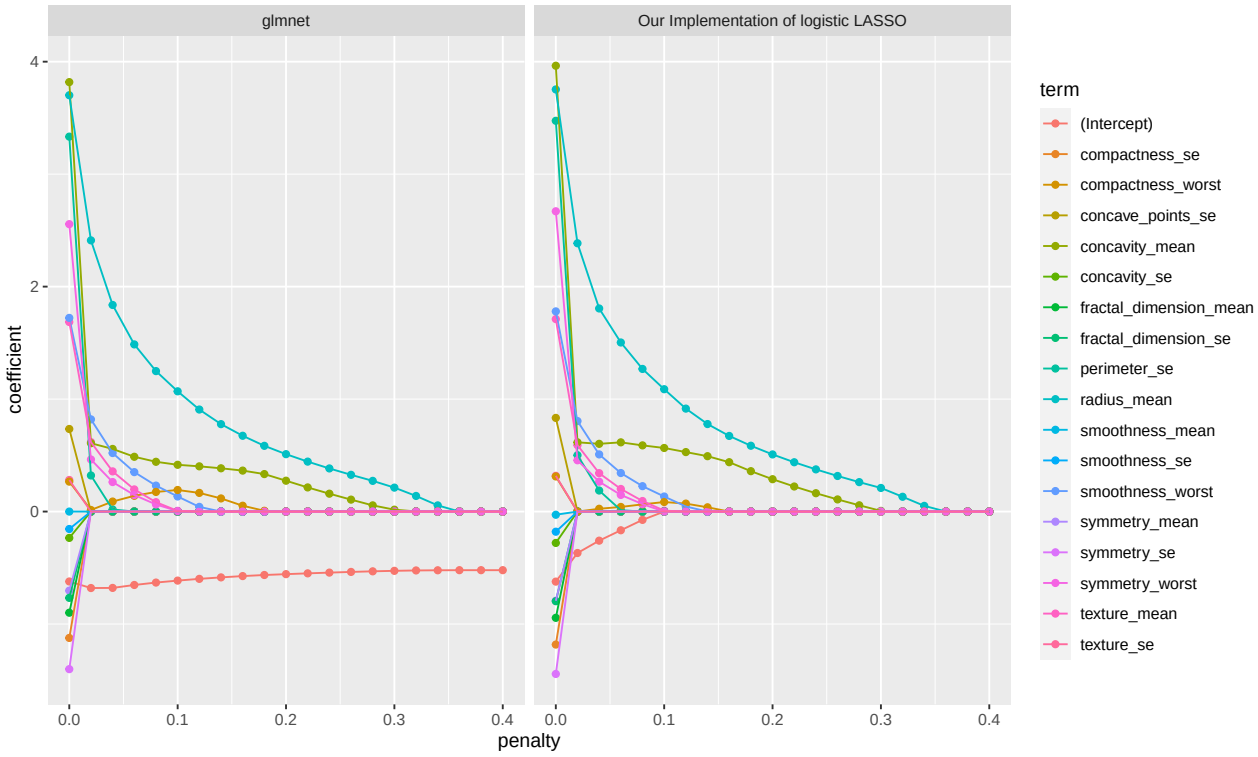
```
## [1] 0.3725835
```

With $\lambda_{max} = max_l \frac{<X_l Y>}{N} = 0.3725835$, we make a sequence from 0 to 0.4, by 0.02 per step and make path plots.

```
lambda_seq<-seq(from = 0, to = 0.4, by = 0.02)

path_lasso_custom<-function(my_seq){
        for_map<-function(x){
                res_1<-logistic_lasso(data=breast_int_x,
                      y = breast_int_y,
                      lambda = x,
                      beta_vec = as.matrix(rep(0,ncol(breast_int_x)),ncol = 1),
                      beta_name = c(names(breast_int_x)
                                    [which(names(breast_int_x) != "diagnosis")]))
                res_2<- res_1$estimates %>% mutate(penalty = x,
                                                   func = "logistic_lasso")
        }
        res<-map(my_seq,for_map) %>% reduce(rbind) %>%
                mutate(term = ifelse(term == "intercept", "(Intercept)",term))
        res
}

path_lasso_glmnet<-function(my_seq){
        test_dat <- breast_no_int %>%
                        mutate(diagnosis  = factor(diagnosis , levels = c(0, 1)))

                my_rec <- recipe(diagnosis  ~ ., data = test_dat)

        for_map<-function(x){
                my_model <- logistic_reg(penalty = x, mixture = 1) %>%
                        set_engine("glmnet") %>%
                        set_mode("classification")

                wf <- workflow() %>%
                        add_recipe(my_rec) %>% # Workflow
                        add_model(my_model)

                # Compare the betas of the model with the true beta
                fit_data <- wf %>%
                        fit(test_dat) %>%
                        pull_workflow_fit() %>% tidy()
                fit_data
        }
        res<-map(my_seq,for_map) %>% reduce(rbind) %>%
                rename(coefficient = estimate)
        res$func = "glmnet"
        res
}

path_my_function<-path_lasso_custom(lambda_seq)
path_glm_net<-path_lasso_glmnet(lambda_seq)

for_viz<- rbind(path_my_function, path_glm_net) %>%
```

```
        mutate(func = if_else(func == "logistic_lasso",
                              "Our Implementation of logistic LASSO", func))
```

```
for_viz %>% ggplot(aes(x = penalty, y = coefficient,color = term))+
        geom_point(aes())+
        geom_line(aes(group = term))+
        facet_wrap(~func)
```



This figure shows us the path of coefficients of increasing λ between `{glmnet}` and our implementation. Our function regularizes the coefficient term, while `{glmnet}` does not.

4.2.3 Cross-Validation to find optimal penalty

We perform a 10 fold CV with 5 repeats each to tune our λ. We select the best λ based on best ROC AUC. Our best λ is 0.00475. We compare our function to `{glmnet}` at $\lambda = 0.00475$.

```
my_model <- logistic_reg(penalty = tune(), mixture = 1) %>%
        set_engine("glmnet") %>%
        set_mode("classification")

ten_fold<- vfold_cv(test_dat, repeats = 5, strata = "diagnosis")

reg_grid <- grid_regular(penalty(), levels = 100)
```

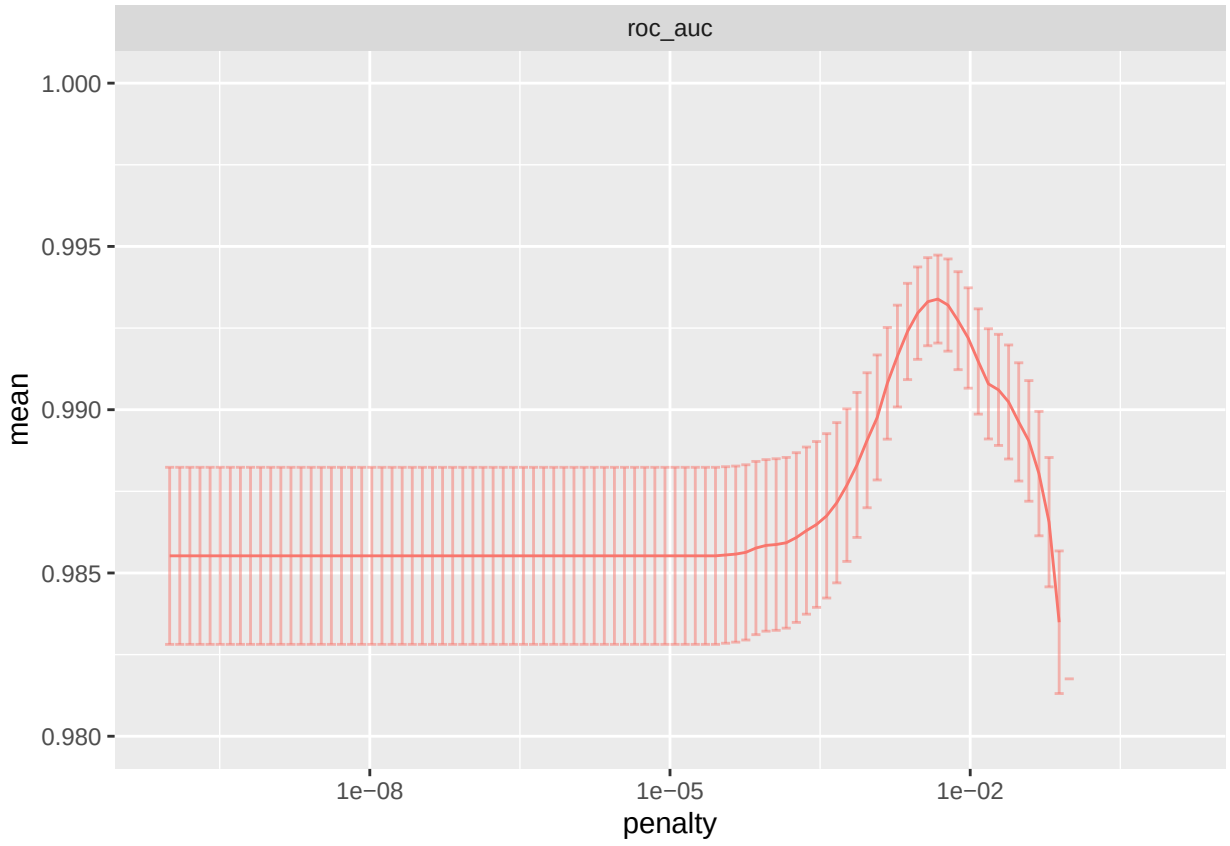
```
## Tune the model and pick model based on best roc_AUC
wf <- workflow() %>%
        add_recipe(my_rec) %>% # Workflow
        add_model(my_model)

## Parallel
all_cores <- parallel::detectCores(logical = FALSE)-1
cl <- parallel::makePSOCKcluster(all_cores)
doParallel::registerDoParallel(cl)

tune_lasso_grid <- tune_grid(
        wf,
        ten_fold,
        grid = reg_grid
)

parallel::stopCluster(cl)
```

4.2.3.1 Using grid search to tune lasso

```
tune_lasso_grid %>%
        collect_metrics() %>% filter(`.metric` == "roc_auc") %>%
        ggplot(aes(penalty, mean, color = `.metric`)) +
        geom_errorbar(aes(
                ymin = mean - std_err,
                ymax = mean + std_err),
                alpha = 0.5) +
        geom_line(size = 0.5) +
        scale_x_log10() +
        facet_wrap(~ `.metric`, scale = "free")+
        theme(legend.position = "none")+
        ylim(0.98,1)
```

4.2.3.2 Select the best penalty term

```
## Warning: Removed 11 row(s) containing missing values (geom_path).
```




```
min_auc <- tune_lasso_grid %>%
        select_best("roc_auc")
final_wf <- finalize_workflow(
        wf,
        min_auc
)

fit_data <- final_wf %>%
        fit(test_dat) %>%
        pull_workflow_fit() %>% tidy()

our_function<-path_lasso_custom(0.00475) %>% select(-func,-penalty) %>%
        rename("our_function" = coefficient) %>%
        left_join(fit_data, by = "term") %>%
        rename("glmnet" = estimate)

our_function %>% knitr::kable() %>%
        kable_classic_2(latex_options = "HOLD_position", position = "center")
```

term	our_function	glmnet	penalty
(Intercept)	-0.4616131	-0.6290945	0.0047508
radius_mean	2.9808754	2.9772860	0.0047508
texture_mean	1.0780735	1.0957242	0.0047508
smoothness_mean	0.0000000	0.0000000	0.0047508
concavity_mean	1.6684822	1.6445506	0.0047508
symmetry_mean	0.0000000	0.0000000	0.0047508
fractal_dimension_mean	-0.1354466	-0.1629368	0.0047508
texture_se	0.0000000	0.0000000	0.0047508
perimeter_se	1.5736338	1.4535666	0.0047508
smoothness_se	0.0000000	0.0000000	0.0047508
compactness_se	-0.4348508	-0.3553487	0.0047508
concavity_se	0.0000000	0.0000000	0.0047508
concave_points_se	0.0000000	0.0000000	0.0047508
symmetry_se	-0.4557500	-0.5011653	0.0047508
fractal_dimension_se	-0.2679371	-0.2947295	0.0047508
smoothness_worst	1.1571920	1.1613804	0.0047508
compactness_worst	0.0000000	0.0000000	0.0047508
symmetry_worst	1.0114985	1.0351161	0.0047508

5 Summary

- Our function regularizes the intercept, whereas `{glmnet}` doesn't.
- The estimates are very similar, indicating that our code is correct.
- `radius_mean` seems to have the highest beta. This means that at each point increase of standard deviation in `radius_mean`, the odds of being a "M" cell increases by 2.98 times.

6 Reference

- Hastie, T. J., Tibshirani, R. J., & Wainwright, M. J. (2015). Statistical learning with sparsity: The lasso and generalizations. Boca Raton: CRC Press/Taylor & Francis.
- Silge, J. LASSO regression using tidymodels and #TidyTuesday data for The Office. https://juliasilge.com/blog/lasso-the-office/./