Assignment 2

Chengshuo Zhang

11/27/2020

Task of the assignment

computing the logistic Lasso and penalizing iteratively reweughted least square algorithm by coordinate descent

Introduction

To use the code in this vignette, you will need to install the packages: tidymoels and tidyverse.

the following two functions are added to a `parsnip` model to be used in `tidymodels` workflow.

`ret <- fit_logistic_lasso(x, y, lambda, beta0 = NULL, eps = 0.0001, iter_max = 100)`: we fit logistic regression with with Lasso (l^1 penalty). The detailed algorithm of Lasso logstic regression will be presented in the basic usage.

Input:

- x: matrix of predictors which does not include the intercept
- y: vector of data – response variable
- lambda: penalty parameter
- beta0: initial guess for optimization
- eps: parameter for stopping criterion
- iter_max: maximum number of iterations

Output:

- List of intercept, beta, lambda, factor levels, if convergedand the numebr of iteration when the training is finished (converged or reach the max number of iterations)

`ret <- predict_logistic_lasso(object, new_x)` : we make prediction for new observations by using fitted logistic Lasso regression.

Input:

- object: Output from fit_logistic_lasso – List of intercept, beta, and lambda for the lasso logistic model
- new_x: Data to predict at

Output:

- Predicted values

Basic Usage

`ret <- fit_logistic_lasso(x, y, lambda, beta0 = NULL, eps = 0.0001, iter_max = 100)`: we use use IRLS and coordinate descent to fit the model. the loss function step m in IRLS is

$$(\beta_0^{(m+1)}, \beta^{(m+1)}) = \arg\min_{\beta 0, \beta} \sum_{i=1}^{n} w_i^{(m)} (z_i^{(m)} - \beta_0 - x_i^T \beta)^2 + \lambda|\beta|$$

$$w_i^{(m)} = p^{(m)}(x_i)(1 - p^{(m)}(x_i))$$

$$z_i^{(m)} = (\frac{y_i - p^{(m)}(x_i)}{p^{(m)}(x_i)(1 - p^{(m)}(xi))+} + x_i^{(T)} \beta^{(m)})$$

$$p^{(m)} = \frac{1}{1 + exp(-x_i^T \beta^{(m)} - \beta_0^{(m)})}$$

this function is different from what `Glmnet` uses for Lasso logistic regression.
we use coordinate descent to find $(\beta_0^{(m+1)}, \beta^{(m+1)})$

Now, we are going to show how to treat the data in tidymodels interface for creating the Lasso logsitic model.

First, we use the data set up same as the week_9 tutorial. x is that we evenly take 1000 numbers from [-3, 3], w is that we evenly take 1000 numbers from [-3π, 3π]. y is that we choose 1000 random numbers from binomial distribution where $p_i = 1/(1+exp(-w_i+2x_i))$. cat is a vector with length 1000 of categorical variable which randomly taken from a, b, and c. y is independent from cat. Then, we create a simple training/test set split. Then we specify formular for the regression model and processing predictors(normalizing all the numerical predictors) by `recipe`.

```
source("functions.R")
source("make_tidy.R")
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.1.1 --

## v broom     0.7.2      v recipes   0.1.14
## v dials     0.0.9      v rsample   0.0.8
## v dplyr     1.0.2      v tibble    3.0.4
## v ggplot2   3.3.2      v tidyr     1.1.2
## v infer     0.5.3      v tune      0.1.1
## v modeldata 0.1.0      v workflows 0.2.1
## v parsnip   0.1.4      v yardstick 0.0.7
## v purrr     0.3.4

## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()

## Information for `IRLS`
##  modes: unknown, classification
##
##  engines:
##    classification: fit_logistic_lasso
##
##  no registered arguments.
```

```
##
##  no registered fit modules.
##
##  no registered prediction modules.
```

```
set.seed(789)
n = 1000
dat <- tibble(x = seq(-3,3, length.out = n),
              w = 3*cos(3*seq(-pi,pi, length.out = n)),
              y = rbinom(n,size = 1, prob = 1/(1 + exp(-w+2*x)) )%>% as.numeric %>% factor,
              cat = sample(c("a","b","c"), n, replace = TRUE)
)
split <- initial_split(dat, strata = c("cat"))
train <- training(split)
test <- testing(split)
rec <- recipe(y ~ . , data = train) %>%
  step_dummy(all_nominal(), -y) %>% step_zv(all_outcomes()) %>%
  step_normalize(all_numeric(), -y)  # don't normalize y!
```

we specify model with λ and engine as `spec`. The engine is the "fit_logisticv_lasso". Then ,we fit the model with training data and predict testing data using fitted model. the `workflow` in fit process data use `rec`, and then use the model in `spec`, and in the end fit model with training date. for `predict`,we use the same data processing as fit. The underlying function for fit is `ret <- fit_logistic_lasso(x, y, lambda, beta0 = NULL, eps = 0.0001, iter_max = 100)` and the underlying function for predict is `ret <- predict_logistic_lasso(object, new_x)`. we presented confusion matrix and beta.

```
spec <- IRLS(lambda=100) %>% set_engine("fit_logistic_lasso")
fit <- workflow() %>% add_recipe(rec) %>% add_model(spec) %>% fit(train)
predict(fit, new_data = test) %>% bind_cols(test %>% select(y)) %>%
  conf_mat(truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0  93   9
##          1  18 129
```

```
fit$fit$fit$fit$beta
```

```
##         x         w     cat_b     cat_c
## -1.586775  0.564068  0.000000  0.000000
```

now we change the values of λ to compare the results.

```
spec <- IRLS(lambda=50) %>% set_engine("fit_logistic_lasso")
fit <- workflow() %>% add_recipe(rec) %>% add_model(spec) %>% fit(train)
predict(fit, new_data = test) %>% bind_cols(test %>% select(y)) %>%
  conf_mat(truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0 104  15
##          1   7 123
```

```
fit$fit$fit$fit$beta
```

```
##          x          w      cat_b      cat_c
## -2.1448688  0.9735513  0.0000000  0.0000000
```

```
spec <- IRLS(lambda=25) %>% set_engine("fit_logistic_lasso")
fit <- workflow() %>% add_recipe(rec) %>% add_model(spec) %>% fit(train)
predict(fit, new_data = test) %>% bind_cols(test %>% select(y)) %>%
  conf_mat(truth = y, estimate = .pred_class)
```

```
##            Truth
## Prediction   0   1
##          0 105  15
##          1   6 123
```

```
fit$fit$fit$fit$beta
```

```
##         x         w     cat_b     cat_c
## -2.637961  1.320796  0.000000  0.000000
```

From the above three λ values, we can see that they all exclude the unrelated variable cat. when $\lambda = 100$, the x value is a liitle bit apart from -2. since the two other $\lambda = 50, 25$ can exclude the unrelated variable cat, $\lambda = 100$ is too large. we can use cross validation to choose a proper λ.