Vignette

Mahrukh Niazi 1003948204

12/4/2020

Overview:

Logistic lasso is a predictive modeling problem where a class label is predicted based on a given input of data. This vignette covers how to compute logistic lasso using coordinate descent on the penalised iteratively reweighted least squares algorithm, then add the model as a `parsnip` model so that it can be used in a `tidymodels` workflow to perform classification. After creating the model and using it within the `tidymodels` workflow, this vignette will then show how to tune the model and find the best model.

Getting started:

First, we load `tidyverse` and `tidymodels` as they are required for the logistic model to run. Then, we call `source("functions.R")` and `source("make_tidy.R")` which contain the code that fits and predicts the logistic lasso model and registers it as a `parsnip` model.

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

Deriving the fit and predict functions:

Note: the functions `fit_logistic_lasso` and `predict_logistic_lasso` are in the `functions.R` file.

The function `fit_logistic_lasso` is used to minimize the log-likelihood function of the lasso penalised logistic regression:

$$\log L_p(\boldsymbol{y}, \boldsymbol{\beta}) = \sum_{i=1}^{n} -[y_i \log(\pi_i) + (1 - y_i)\log(1 - \pi_i)] + \lambda \sum_{j=1}^{p} |\beta_j|$$

where the parameter $\hat{\beta}$ is estimated to be:

$$\hat{\beta}_j = \frac{S(\frac{1}{n}\sum_{i=1}^{n} w_i x_{ij} r_i + (\frac{1}{n}\sum_{i=1}^{n} w_i x_{ij}^2)\beta_{j_{est}}, \lambda\alpha)}{\frac{1}{n}\sum_{i=1}^{n} w_i x_{ij}^2 + \lambda(1-\alpha)}$$

where $w_i = \pi_{i_{est}}(1 - \pi_{i_{est}})$, $r_i = y_i - \beta_{0_{est}} - x_i^T \beta_{est}$, and $\beta_{j_{est}}$ is the current estimate of β_j.

$$S(c, \gamma) = \begin{cases} c - \gamma, & if\ c > 0, and\ \gamma < |c| \\ c + \gamma, & if\ c < 0, and\ \gamma < |c| \\ 0, & if\ \gamma \geq |c| \end{cases}.$$

The $\hat{\beta}_j$ uses a soft thresholding function defined as:

`fit_logistic_lasso` takes a matrix of predictors (not including the intercept) that has been normalised `x`, a vector of target binary data `y`, a vector of initial beta guess values `beta0` which is set to `NULL` as default, a penalty parameter `lambda`, a stopping criterion `eps` (set to `0.0001` as default), and a maximum number of iteration value `max_iter` (set to `100` as default) as inputs. It returns a list of beta and intercept estimates that will be used to make predictions and classify the data points using the test set of the data.

The function `predict_logistic_lasso` is used to make predictions for the classifications of the data points. It takes the output of `fit_logistic_lasso` and the x values of the test data set `new_x` as inputs and returns a list of the intercept and beta estimates for the new x values.

Adding model as parsnip model:

Once the `fit_logistic_lasso` and `predict_logistic_lasso` functions have been implemented, they can be added as a `parsnip` model so that it can be used in a `tidymodels` workflow. This is done through a series of steps. First, the classification model is registered using the function `logistic_lasso_IRLS` which takes `mode = "classification` and `penalty` as inputs, where `penalty` is the lambda penalization parameter that is needed in `fit_logistic_lasso`. The `logistic_lasso_IRLS` model is set using `set_new_model("logistic_lasso_IRLS")` and specifying `mode = "classification` and `eng = "fit_logistic_lasso"` in `set_model_model` and `set_model_engine` respectively. Then, `set_model_arg` is used to tell `parsnip` what parameters are used in the model: `fit_logistic_lasso` uses lambda as a parameter. `set_encoding` is used to tell `parsnip` to treat factors the same way `lm()` does by setting the argument as `predictor_indicators = "traditional"` and remove the intercept column from the matrix using `remove_intercept = TRUE` as argument. Lastly, the fitting and prediction models are registered in `parsnip` using `set_fit` and `set_pred`. The engine used for both `set_fit` and `set_pred` is `eng = "fit_logistic_lasso"` and the function `func` argument is specified as `fit_logistic_lasso` in set_fit and `predict_logistic_lasso` in `set_pred`. An update function `update.IRLS` is used to update the model with the final parameter.

Test model's accuracy:

To test our logistic lasso model's accuracy, we examine its confusion matrix. A confusion matrix is a table that summarises the performance of a classification model on a set of test data. To do this, we first generate a random data set, clean it, and split it into a training set and a test set. The training set is used to fit the model whereas the test set is used to evaluate the model fit. A test set is needed to avoid overfitting and provide better estimates. After generating data, we set the engine of the `logistic_lasso_IRLS` model in `make_tidy.R` to the fit function `fit_logistic_lasso` in `functions.R` and train the model using the training set. Lastly, we use the `predict()` function to generate a confusion matrix for the `logistic_lasso_IRLS` model using the test set.

This is demonstrated below:

```r
# Generate random data:
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 data into training set and test set:
split <- initial_split(dat, strata = c("cat"))
train <- training(split)
test <- testing(split)

# Normalize all predictor variables and convert categorical variables to binary dummy variables:
rec <- recipe(y ~ . , data = train) %>%
  step_dummy(all_nominal(), -y) %>% step_zv(all_outcomes()) %>%
  step_normalize(all_numeric(), -y) # don't normalize y!
```

Set logistic_lasso_IRLS model with penalty = 0.3 and add to tidymodels workflow:
```
# Set logistic_lasso_IRLS model with penalty = 0.3 and add to tidymodels workflow:
spec <- logistic_lasso_IRLS(penalty = 0.3) %>% 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 97 27
##          1 31 95
```

The confusion matrix shows:

- The model predicted 0 and actually got 0 110 times. This meands the model had 110 true negatives.
- The model predicted 0 but actually got 1 23 times. This means the model had 23 false negatives.
- The model predicted 1 and actually got 1 91 times. This means the model had 91 true positives.
- The model predicted 1 but actually got 0 25 times. This means the model had 25 false positives.

Evidently, the model is slightly more likely to have false positives than false negatives. The misclassification rate is given by:

$$\frac{(23+25)}{110+23+25+91} = \frac{48}{249} \approx 19.23\%$$

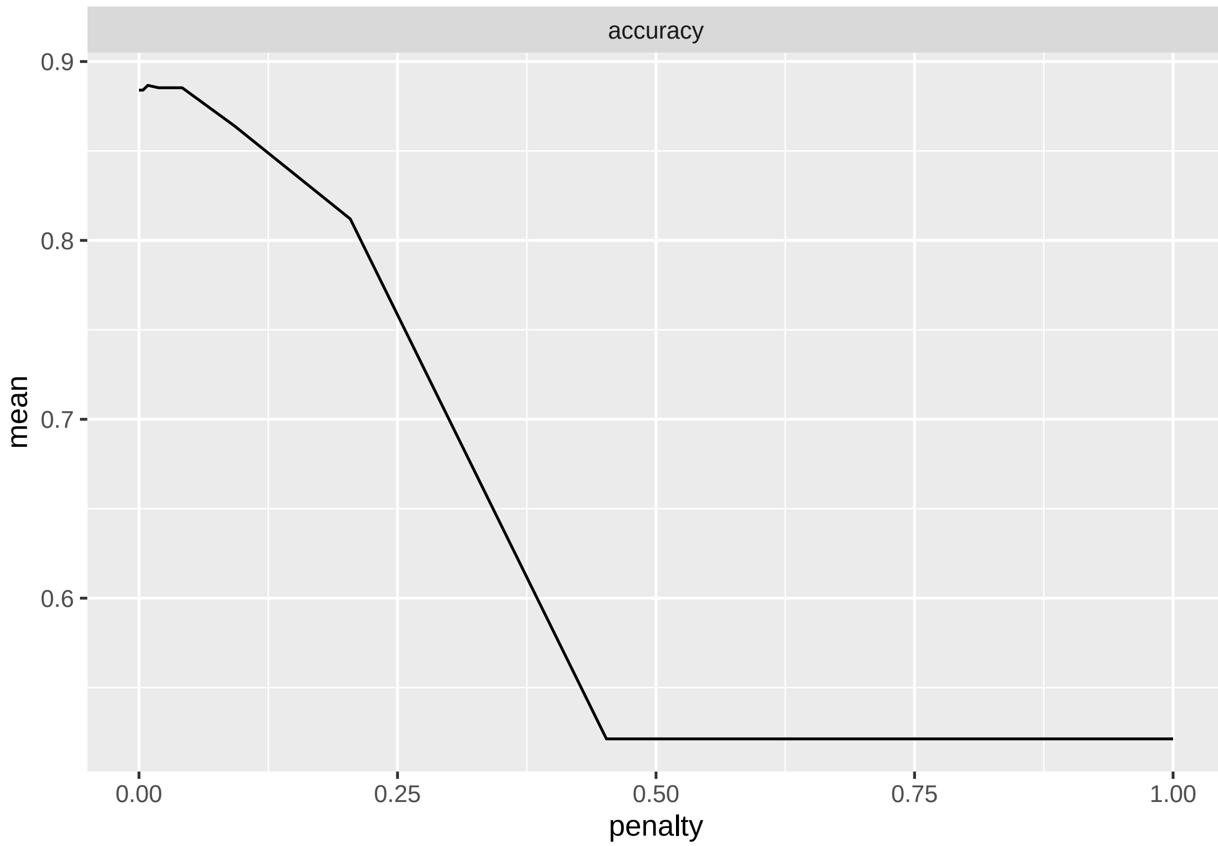
Now, we want to see if we can reduce the misclassification rate by tuning the model so the model performs better.

Tune the Model

The model is tuned to estimate the best values of the parameters (betas) by training the model on resampled data sets and evaluating how well the model perform. Using `grid_regular`, we create grids of tuning parameters as a way to efficiently train many models using resampled data. Setting `levels = 30`, we test 30 values of lambda. Next, we specify the penalty parameter as the parameter that will be tuned. So we tune the `logistic_lasso_IRLS` penalty and add it to the workflow. Then, we perform cross validation on the training sets so we can choose the best value for lambda. Finally, the fitted model is tuned and saved as `fit_tune`.

```
set.seed(1)

grid <- grid_regular(penalty(), levels = 30)
spec_tune <- logistic_lasso_IRLS(penalty = tune()) %>% set_engine("fit_logistic_lasso")
wf <- workflow() %>% add_recipe(rec) %>% add_model(spec_tune) #combines model specification and preproc
folds <- vfold_cv(train) #perform cross validation on the training data
fit_tune <- wf %>% tune_grid(resamples = folds, grid = grid, metrics = metric_set(accuracy))
fit_tune %>% collect_metrics() %>% ggplot(aes(penalty, mean)) + geom_line() + facet_wrap(~.metric)
```



From the graph we see that accuracy plateaus to a penalty level of around 0.50. This indicates a smaller penalty term may be better for the model performance.

Now we use the `select_best` function on `fit_tune` to pull out the best parameter (lambda) values the model:

```
set.seed(1)

penalty_final <- fit_tune %>% select_best(metric = "accuracy")

# Update workflow object wf with values from select_best and add to workflow:
wf_final <- wf %>% finalize_workflow(penalty_final)
```

Looking at the finalised workflow object, we see the penalty value `1e-10` that is suggested after tuning:

```
set.seed(1)

wf_final
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: logistic_lasso_IRLS()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_dummy()
## * step_zv()
## * step_normalize()
```

```
##
## -- Model ----------------------------------------------------------------------
## Model Specification (classification)
##
## Main Arguments:
##   penalty = 0.00853167852417281
##
## Computational engine: fit_logistic_lasso
```

Finally, the final model, with the best lambda, can be fitted to the training data. Using `predict()`, we can examine the confusion matrix and compare with before:

```r
set.seed(1)

final_fit <- wf_final %>% fit(train)
predict(final_fit, new_data = test) %>%
  bind_cols(test %>% select(y)) %>%
  conf_mat(truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0 116  18
##          1  12 104
```

The confusion matrix shows:

- The model predicted 0 and actually got 0 121 times. This meands the model had 121 true negatives.
- The model predicted 0 but actually got 1 12 times. This means the model had 12 false negatives.
- The model predicted 1 and actually got 1 102 times. This means the model had 102 true positives.
- The model predicted 1 but actually got 0 14 times. This means the model had 14 false positives.

The misclassification rate is given by:

$$\frac{(12+14)}{121+12+14+102} = \frac{26}{249} \approx 10.44\%$$

As we can see, after tuning the model the misclassfication rate decreased to 10.44%. Therefore, the tuned model performs better.

Summary:

Throughout this vignette, it was demonstrated how to compute logistic lasso using coordinate descent on the penalised iteratively reweighted least squares algorithm, add the model to the `tidymodels` workflow, tune the model, select the best penalty term, and use the confusion matrix to determine if the model performs well. Note, logistic models are not perfect and with more complicated data a more complicated model may be needed.