Logistic Regression: Feature Selection and Cross Validation, Part 3

Will Doyle

2022-07-12

Today we're going to continue or modeling approach by using a model designed to select those features which are most closely associated with a student yielding, and dropping correlated variables that don't contribute as much to the model. We'll validate our approach using cross validation, just like we did in the context of regression.

```
library(modelr)
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'
```

```
library(tidyverse)
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'hms'
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
library(tidymodels)
```

```
## Warning: package 'recipes' was built under R version 4.1.2
```

```
library(knitr)
```

```
ad<-read_rds("admit_data.rds")
```

Data Wrangling

When setting up our dependent variable, we need to specify the reference category. This is very important and affects how we interpret the results of the model once we have them. I'm also going to make some adjustments, measure SAT in 100s and income and distance in 1000s. This will make it easier to understand our estimates.

```
ad_sub <- ad %>%
  mutate(yield_f = as_factor(ifelse(yield == 1, "Yes", "No"))) %>%
  mutate(yield_f = relevel(yield_f, ref = "No")) %>%
  mutate(sat=sat/100,
         income=income/1000,
         distance=distance/1000)%>%
  select(ID,
         yield_f,
         legacy,
         visit,
         registered,
         sent_scores,
         sat,
```

```
        income,
        gpa,
        distance,
        net_price)
```

Settting the Model

We're going to run a LASSO model again, this time using the logit link function. To do this, we need to use the `logistic_reg` specification for a `glmnet` model. Note that this is just like what we did in regression, except we're specifying a logistic regression for classification as opposed to a linear model.

Remember that the lasso downweights variables mostly by dropping variables that are highly correlated with one another, leaving only one of the correlated variables as contributors to the model. We set the degree to which this penalty will be implemented by setting the "penalty" variable in the model specification.

```
penalty_spec<-.1

mixture_spec<-1

lasso_logit_mod<-
  logistic_reg(penalty=penalty_spec,
               mixture=mixture_spec) %>%
  set_engine("glmnet")%>%
  set_mode("classification")
```

Formula and Recipe

We'll set up a formula that includes all of the variables in the dataset. Notice that because we're using the lasso model, I will go ahead and normalize all of the predictors, turning them into Z scores.

```
admit_formula <- as.formula(
                         "yield_f~.")

admit_recipe<-recipe(admit_formula,ad_sub)%>%
  update_role(ID, new_role = "id variable")%>%
  step_log(distance)%>%
  step_normalize(all_predictors())%>%
  step_naomit(all_predictors())
```

Workflow

Having set up the model and the recipe, I can create a workflow.

```
ad_wf<-workflow()%>%
  add_model(lasso_logit_mod)%>%
  add_recipe(admit_recipe)
```

Set Resamples

To properly evaluate model fit, we need to resample as always. We'll use a Monte Carlo resample. I'm using 25 resamples for this example, but in practice we'd want to do more, 1000 resamples would be typical in

many applications.

```
ad_rs<-mc_cv(ad_sub,times=25) # DO MORE IN PRACTICE
```

Fit the Model: THIS WILL TAKE A FEW SECONDS, JUST BREATHE . . .

```
options(yardstick.event_first = FALSE)

ad_lasso_fit<-ad_wf%>%
  fit_resamples(ad_rs,  metrics = metric_set(roc_auc, sens, spec,accuracy))
```

```
## Warning: package 'rlang' was built under R version 4.1.2
```

```
## Warning: package 'vctrs' was built under R version 4.1.2
```

```
## Warning: package 'glmnet' was built under R version 4.1.2
```

The code above fits the model to the resampled dataset, generating parameter estimates from the training split and estimates of predictive accuracy from the testing split.

Check Accuracy of the Model

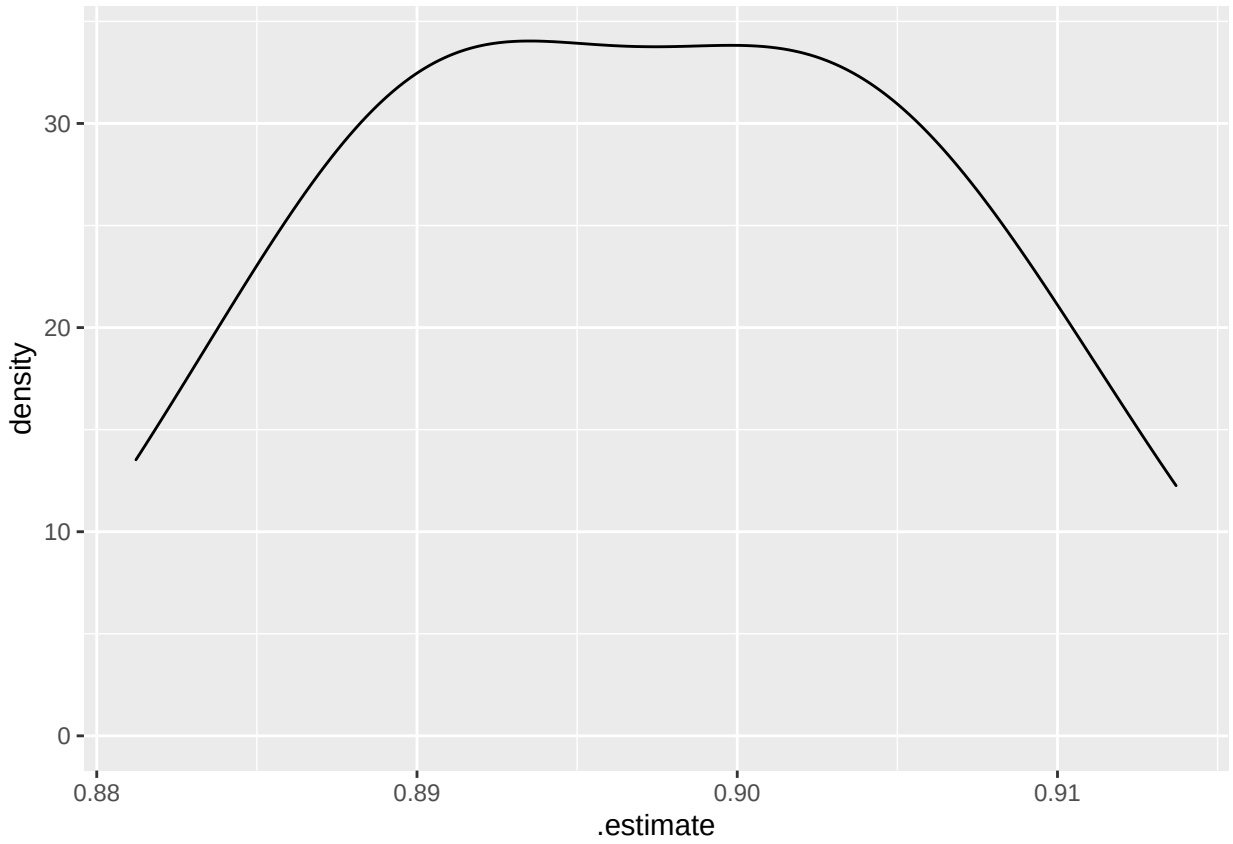
We can use the `collect_metrics` command to get the average accuracy and AUC from our model.

```
ad_lasso_fit%>%
  collect_metrics()
```

```
## # A tibble: 4 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.722    25 0.00382 Preprocessor1_Model1
## 2 roc_auc  binary     0.897    25 0.00182 Preprocessor1_Model1
## 3 sens     binary     0.161    25 0.00903 Preprocessor1_Model1
## 4 spec     binary     0.986    25 0.00179 Preprocessor1_Model1
```

We can also plot the auc for every resampled version:

```
ad_lasso_fit%>%
  unnest(.metrics)%>%
  filter(.metric=="roc_auc")%>%
  ggplot(aes(x=.estimate))+
  geom_density()
```



This shows us that in general this model has an AUC of .9, which is fairly high. At it's lowest it's around .8, and it maxes out close to 1.

Quick Exercise: Plot the Accuracy of the model across resamples

```
# INSERT CODE HERE
```

Finalize Workflow

We can now select the best model fit from our resamples, and apply it to the full dataset.

```
ad_lasso_final <- finalize_workflow(ad_wf,                                    select_best(ad_lasso_fi
  fit(ad_sub)
```

Parameter Estimates

We can then examine the parameter estimates from the model run on the full dataset.

```
ad_lasso_final%>%
  extract_fit_parsnip()%>%
  tidy()%>%
  kable()
```

term	estimate	penalty
(Intercept)	0.8791537	0.1
legacy	0.0000000	0.1
visit	0.0000000	0.1

term	estimate	penalty
registered	0.0000000	0.1
sent_scores	0.0000000	0.1
sat	0.2278358	0.1
income	0.7147746	0.1
gpa	0.0000000	0.1
distance	0.0000000	0.1
net_price	0.0000000	0.1

With the very high penalty, the model is selecting just sat and income as the best predictors of yield.

Hyperparameter Tuning

There's no guarantee we chose the right penalty. Let's see what difference it makes by using hyperparameter tuning to allow the penalty to vary across a sensible range of possibilities.

Set Up Lasso Model

To set up the model for tuning, we're now setting `penalty=tune()`.

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

Set Grid

Using the `grid_regular` command will give us a reasonable set of possible penalties to work with.

```
lasso_grid<-grid_regular(parameters(lasso_logit_mod) ,levels=10)
```

Update Model

With this new specification, we can update the workflow.

```
ad_wf<-ad_wf%>%
  update_model(lasso_logit_mod)
```

Fit Using the Grid: THIS WILL TAKE A WHILE; ENHANCE YOUR CALM . . .

Now we can fit the model to the data, fitting each of the possible 10 penalty to values to each of the resampled datasets.

```
ad_tune_fit <-
  ad_wf %>%
    tune_grid(ad_rs,grid=lasso_grid, metrics = metric_set(roc_auc, sens, spec,accuracy))
```

Show AUC and other measures of fit in resamples

Having fit the model, we can take a look at how many different measures of model fit differ when using different penalties.

```
ad_tune_fit%>%
  collect_metrics()%>%
  filter(.metric=="roc_auc")%>%
  arrange(-mean)
```

```
## # A tibble: 10 x 7
##          penalty .metric .estimator  mean     n std_err .config
##            <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.000464      roc_auc binary     0.916    25 0.00158 Preprocessor1_Model07
##  2 0.0000000001  roc_auc binary     0.916    25 0.00159 Preprocessor1_Model01
##  3 0.00000000129 roc_auc binary     0.916    25 0.00159 Preprocessor1_Model02
##  4 0.0000000167  roc_auc binary     0.916    25 0.00159 Preprocessor1_Model03
##  5 0.000000215   roc_auc binary     0.916    25 0.00159 Preprocessor1_Model04
##  6 0.00000278    roc_auc binary     0.916    25 0.00159 Preprocessor1_Model05
##  7 0.0000359     roc_auc binary     0.916    25 0.00159 Preprocessor1_Model06
##  8 0.00599       roc_auc binary     0.915    25 0.00158 Preprocessor1_Model08
##  9 0.0774        roc_auc binary     0.897    25 0.00180 Preprocessor1_Model09
## 10 1             roc_auc binary     0.5      25 0       Preprocessor1_Model10
```

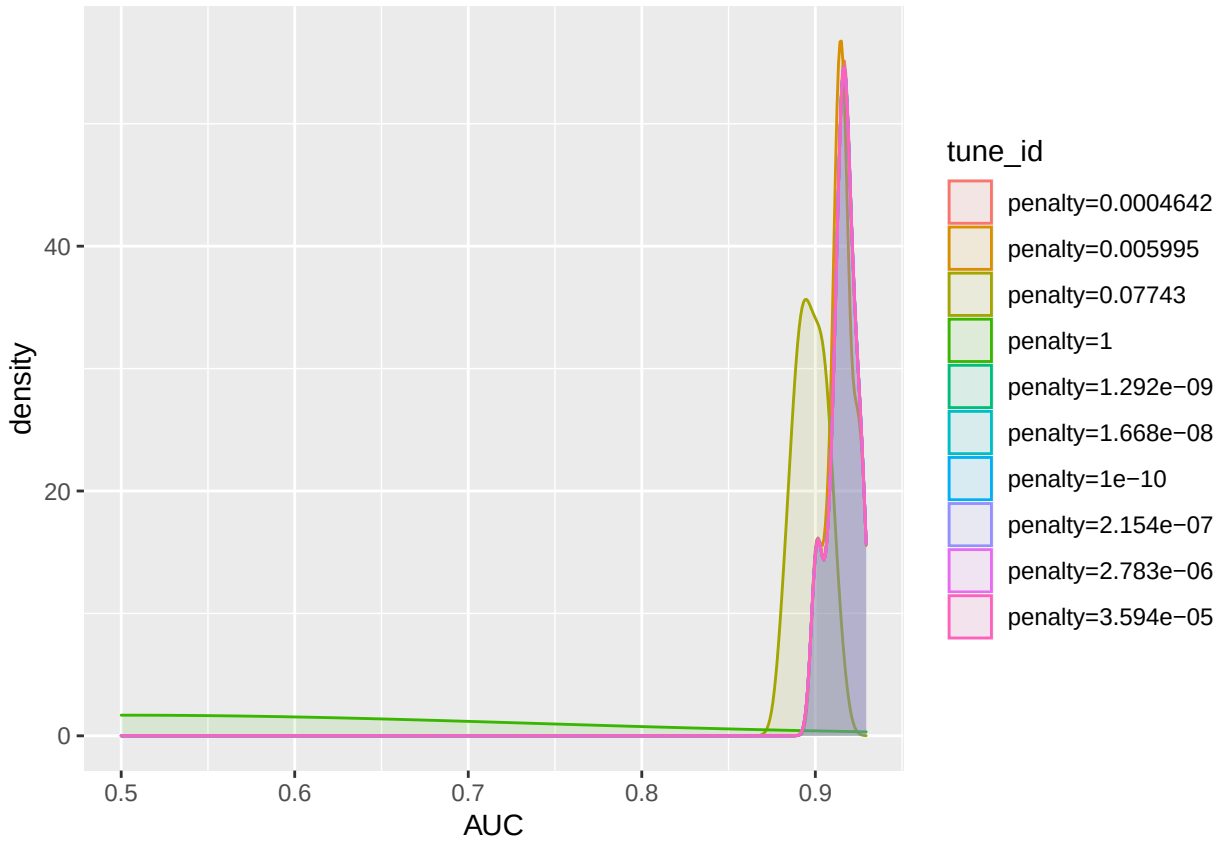
This tells us that smaller penalties are generally yielding better AUC.

Quick Exercise: Output the results for the other measures of model fit (accuracy, sensitivity, specificity). What do these tell you?

```
# INSERT CODE HERE
```

We can plot the results and confirm that suspicion:

```
ad_tune_fit%>%
unnest(.metrics)%>%
  filter(.metric=="roc_auc")%>%
  mutate(tune_id=paste0("penalty=",prettyNum(penalty,digits=4))) %>%
  select(tune_id,.estimate)%>%
  rename(AUC=.estimate)%>%
  ggplot(aes(x=AUC,color=tune_id,fill=tune_id))+
  geom_density(alpha=.1)
```



Choose best model and fit to training data

Having fit the model to resamples and identified the best model fit through cross validation, we can select that model and apply it to the full dataset.

```
ad_final <-
  finalize_workflow(ad_wf,
                    select_best(ad_tune_fit,
                                metric = "roc_auc")) %>%
  fit(ad_sub)
```

The object `ad_final` is a final fitted workflow, which we can use to examine parameter estimates and generate predicted probabilities.

Examine Parameter Estimates

With that model fit in hand, we can check on our parameter estimates.

```
ad_final%>%
  extract_fit_parsnip()%>%
  tidy()%>%
  mutate(penalty=prettyNum(penalty,digits=4))%>%
  kable()
```

term	estimate	penalty
(Intercept)	2.8709837	0.0004642

term	estimate	penalty
legacy	0.2279305	0.0004642
visit	0.1392716	0.0004642
registered	0.2243953	0.0004642
sent_scores	0.2953763	0.0004642
sat	0.0000000	0.0004642
income	5.0013621	0.0004642
gpa	0.4059959	0.0004642
distance	-0.3990855	0.0004642
net_price	-0.8248575	0.0004642

Notice how SAT has been downweighted to 0, while income is a strong predictor.

Next Steps

To improve model fit from here, we could examine the data a bit more carefully and see if there are other ways we might structure it, including other transformations of the variables.

Extension: Predicting from the fitted model

To make out-of-sample predictions from the final fit, we need to specify specific values of the variables of interest. Here I'm going to allow net price to vary from its minimum to its maximum to in 100 steps.

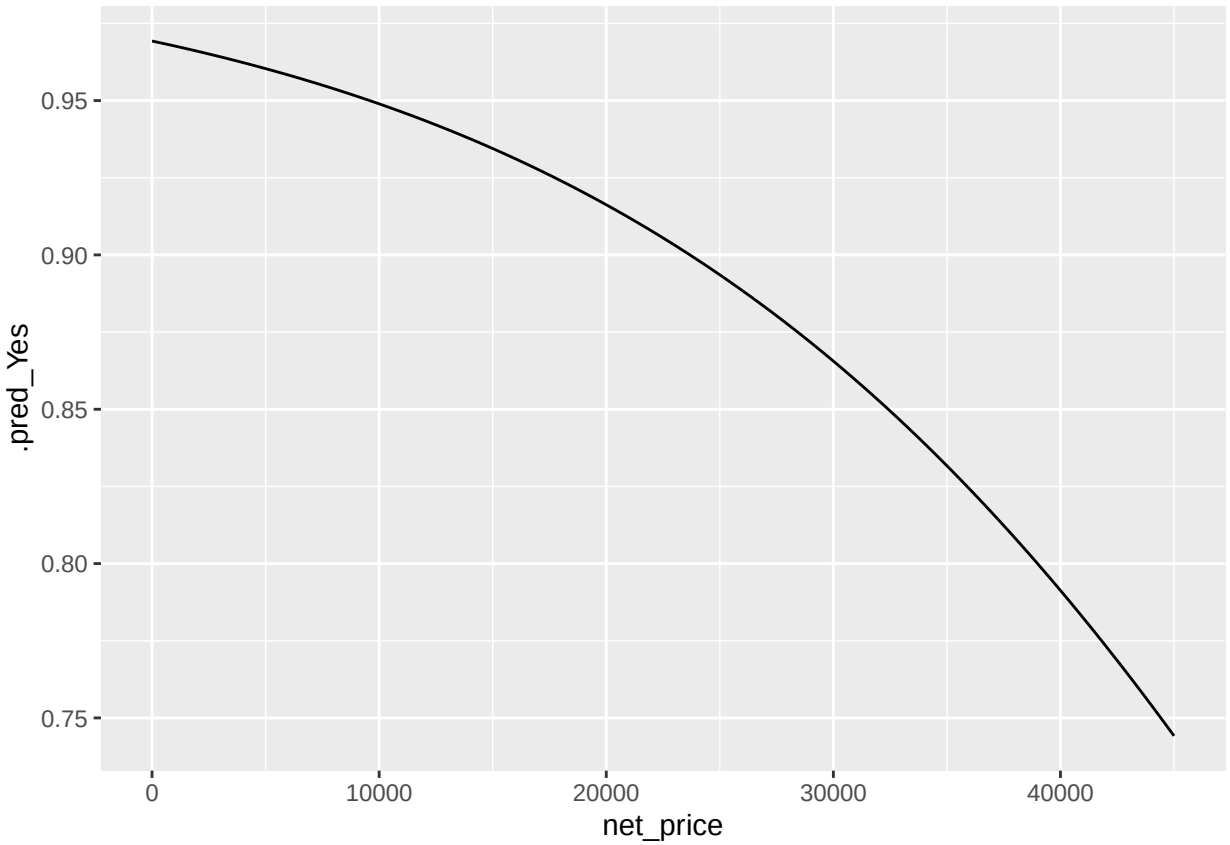
```
hypo_data <- ad %>%
  data_grid(ID="1",
            legacy=0,
            visit=1,
            registered=1,
            sent_scores=1,
            sat=14,
            income=95,
            gpa=3.9,
            distance=.1,
            net_price=seq_range(net_price,n=100)
            )
```

Then I can predict from the final fitted workflow `ad_final`

```
pred_prob<-ad_final%>%
  predict(hypo_data,type="prob")%>%
  bind_cols(hypo_data)
```

... and plot the result

```
pred_prob%>%
  ggplot(aes(x=net_price,y=.pred_Yes))+
  geom_line()
```



Predicting for a change of policy

The above just looks at results for some hypothesized range of inputs. What if we specifically wanted to change policies? Let's add 10,000 dollars to the net price for everyone who sent their scores in.

```
## Add 10k if student sent scores
ad_np<-ad%>%
  mutate(net_price=ifelse(sent_scores==1,net_price+10,net_price))
```

```
pred_np<-ad_final%>%
  predict(ad_np,type="prob")%>%
  bind_cols(ad_np)
```