Variable Selection Methods: Regularization with Ridge and LASSO

August 17, 2024

Table of contents

Recap

With Cross-Validation, we've supercharged our modeling powers. We've made our model performance estimates much more reliable, which gives us greater confidence in understanding the predictive power (or lack-thereof) in our models. We've become much more responsible modelers, in this sense. We'll see again soon that cross-validation can actually do much more for us than just provide more reliable performance estimates than the validation-set approach did. For now though, we shift focus slightly to the question – *how do we know which predictors should be included in our model*?

Motivation

When we first introduced multiple linear regression, we took an approach in which we built a "large" model – a model that included most/all of our available predictors. Once we had that model, we used a procedure called *backward elimination* to eliminate terms from our model if they were associated with p-values above the 5% significance threshold. We could have taken the opposite approach, starting with an empty model, adding predictors/terms one at a time – a process called *forward selection*.

With both of these methods, we're allowing our model to be quite greedy. It may be helpful to think of these processes as allowing our model to go "shopping" for predictors – in the *backward selection* paradigm, the model begins by adding every item in the store to its shopping cart, and then carefully places the ones it doesn't want back on the shelves – in the *forward selection* paradigm, the model begins with an empty cart and adds its favorite items one-by-one into its shopping cart. This seems quite reasonable at first, but we're engaging in quite risky behavior.

- From a purely statistical standpoint, we're evaluating lots of t-tests in determining whether model terms are statistically significant or not. The likelihood of making at least one Type I Error (saying that a model term is statistically significant, when it is not so in the population) becomes very large in these processes.
- From a model-fit perspective, the more predictor variables a model has, the more flexible it is – the more flexible a model is, the better it will fit the *training data* and the more likely it is to become overfit.

By allowing our model to "shop" freely for its predictors, we are enticing our model to overfit the training data – giving our model a "budget" to spend on its shopping trip would be a really nice way to lower the likelihood that our model "buys" too many predictors and overfits the training data.

Optimization Procedures for Model Fitting

All of the models we've fit so far in our course use *Ordinary Least Squares* as the underlying fitting procedure – we are moving to new waters now. Ridge Regression and LASSO models belong to a class of model called *Generalized Linear Models*, although their fitting procedures under the hood will be different, the consistency of the `{tidymodels}` framework allows us to make this change quite simply.

Regularization

The process of applying an additional constraint such as a *budget* one example of model constraints commonly referred to as *regularization*. Regularization techniques are utilized with lots of model classes to help prevent those models from overfitting our training data.

Objectives

After working through this notebook you should be able to:

- Articulate the *backward selection* approach and how it relates to the multiple linear regression models we've constructed and analyzed so far in our course.
- Discuss *forward selection* as an alternative to the backward selection process.
- Discuss and implement *Ridge Regression* and the *LASSO* as alternatives to Ordinary Least Squares within the `{tidymodels}` framework.
- Discuss the benefits and drawbacks for forward- and backward-selection, Ridge Regression, and the LASSO relative to one another.

Ordinary Least Squares

The process of fitting a model using ordinary least squares is an *optimization* problem. Simply put, to fit a model of the form

$$\mathbb{E}\left[y\right] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$$

we minimize the residual sum of squares:

$$\text{Minimize:} \quad \sum_{\text{training data}} \left(y_{\text{observed}} - y_{\text{predicted}}\right)^2$$

Which is equivalent to

$$\text{Minimize:} \quad \sum_{\text{training data}} \left(y_{\text{observed}} - \left(\beta_0 + \sum_{i=1}^{k} \beta_i x_i\right)\right)^2$$

This process of choosing values for $\beta_0,~\beta_1,~\beta_2,~\cdots,~\beta_k$ is called *Ordinary Least Squares* (OLS), and it's what we've been using (or, rather R has been using) all semester long to fit our models.

In OLS, the optimization procedure can freely choose those β values, without any constraint (OLS is shopping without a budget). Ridge Regression and the LASSO introduce extra constraints on this optimization problem.

Ridge Regression

The process of fitting a model using Ridge Regression is similar to that of using OLS, except that a constraint on the chosen β coefficients is imposed. That is, using Ridge Regression to fit a model of the form

$$\mathbb{E}\left[y\right] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$$

Solves the following optimization problem:

$$\begin{array}{ll} \text{Minimize:} & \displaystyle\sum_{\text{training data}} \left(y_{\text{observed}} - \left(\beta_0 + \sum_{i=1}^{k} \beta_i x_i\right)\right)^2 \\ \text{Subject To:} & \displaystyle\sum_{i=1}^{k} |\beta_i| \leq C \end{array}$$

Where C can be thought of as a total coefficient budget. That is, it becomes very expensive for the model to assign lots of non-zero coefficients.

Least Absolute Shrinkage and Selection Operator (LASSO)

The difference between Ridge Regression and the LASSO is in how "spent budget" is calculated. Instead of summing the absolute values of the β coefficients, we'll sum their squares. That is, using LASSO to fit a model of the form

$$\mathbb{E}\left[y\right] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$$

Solves the following optimization problem:

$$\begin{array}{ll} \text{Minimize:} & \displaystyle\sum_{\text{training data}} \left(y_{\text{observed}} - \left(\beta_0 + \sum_{i=1}^{k} \beta_i x_i\right)\right)^2 \\ \text{Subject To:} & \displaystyle\sum_{i=1}^{k} \beta_i^2 \leq C \end{array}$$

Where C can again be thought of as a total coefficient budget. Because it squares the β coefficients in the calculation of "budget used", the LASSO penalizes large coefficients more than Ridge Regression does. Furthermore, due to the mathematics behind these algorithms, models fit using the LASSO result in small coefficients being sent to 0. That is, the LASSO can be used as a *variable selection procedure.* Models fit with Ridge Regression often leave several predictors having very small coefficients so, while they don't have much influence in the overall model predictions, they do still remain in the model.

Some Feature PreProcessing Concerns

If we are going to utilize Ridge Regression or the LASSO, we need to ensure that all of our predictors are on a standardized scale. Otherwise, predictors/features whose observed values are large are superficially *cheaper* for the model to use (because we can attach small coefficients to them) than features whose observed values are already small. Similarly, features whose observed values are very small are artificially made *more expensive* for the model to use because they demand larger coefficients in order to have influence over model predictions. There are two very common scaling methods:

- Min/Max scaling projects a variable onto the interval $[0, 1]$, where the minimum observed value is sent to 0 and the maximum observed value is sent to 1.
  - Min/Max scaling for the variable x is done using the formula: $\dfrac{x - \min(x)}{\max(x) - \min(x)}$.
  - We can add `step_range()` to a `recipe()` to min/max scale a feature.
- Standardization converts a variables raw measurements into standard deviations (z-scores), where the mean of the transformed variable is 0 and the standard deviation of the transformed variable is 1.
  - Standardized scaling for the variable x is done using the formula: $\dfrac{x - \text{mean}(x)}{\text{sd}(x)}$
  - We can add `step_normalize()` to a `recipe()` to standardize the values of a feature.

Fitting a Model Using Ridge Regression or the LASSO

The `tidymodels` framework provides us with a standardized structure for defining and fitting models. Everything we've learned about the syntax and workflow for fitting a linear regression model using OLS can be used to fit these Generalized Linear Models. We'll just need to `set_engine()` to something other than `"lm"` – specifically, an engine which can fit models using the constraints defined in the Ridge Regression and LASSO sections above. We'll use `"glmnet"`, which requires two additional arguments `mixture` and `penalty`.

- The `mixture` argument is a number between 0 and 1. Setting `mixture = 1` results in a LASSO model, while setting `mixture = 0` results in a Ridge Regression model. Values in between 0 and 1 result in a *mixed* LASSO/Ridge approach, where the penalty is partially determined by the Ridge Regression constraint and partially by the LASSO constraint.
- The `penalty` argument corresponds to the amount of *regularization* being applied – that is, the `penalty` is related to the *budget* parameter we've described earlier.

Ummm. . . Great – So What Do I Use and When Do I Use It?

There is some really good news coming – just bear with me!

The choice between Ridge Regression and LASSO depends on your goals. If you are looking for a tool to help with *variable selection*, then the LASSO is your choice, since it will result in less valuable predictors being assigned coefficients of exactly 0. If you are building a predictive model, you might try both and see which one performs better for your specific use-case.

How do I choose a `penalty`? As I understand it, there's no great science to choosing a penalty. People will typically try values and see what the best ones are for their particular use-case.

So basically I've told you about these two new classes of model that help prevent overfitting. They each require a `penalty` parameter and I've given you no real guidance on how to choose it. For now, we'll simply choose a `penalty` parameter and mention what we could build several models with different `penalty` values to try and find one that performs best. I'll give you a much better method when we talk about hyperparameter tuning.

Implementing Ridge Regression and the LASSO

There are a few additional concerns with fitting Ridge Regression and LASSO models that we haven't needed to deal with prior.

- The `glmnet` engine requires that no missing values are included in any fold.
  - We can *omit* rows with missing data. (drawback: we are throwing away observations and we may be drastically reducing the size of our available data)
  - We can *omit* features with missing data. (drawback: we are sacrificing potentially valuable predictors by omitting them from our model)
  - We can *impute* missing values using a `step_impute_*()` feature engineering step in our recipe. (drawback: we are making our best guess at what the missing value should be, but we are introducing additional uncertainty to our model)
- Because the `penalty` parameter imposes a *budget* on coefficients, all of our predictors must be on the same scale – otherwise, some predictors are artificially *cheaper* or *more expensive* than others to include in our model.
  - We can use `step_range()` or `step_normalize()` on `all_numerical_predictors()` to achieve this.

We'll filter out the observations with an unknown response (`Sale_Price`), and we'll use `step_impute_knn()` to use a k-nearest neighbors imputation scheme for any missing values in our predictor columns. Finally, we'll use `step_normalize()` to center and scale our numerical predictors. Then we'll proceed as usual.

Implementing Ridge Regression

We'll build and assess a Ridge Regression model first.

```r
set.seed(123)
ames_known_price <- ames %>%
  filter(!(is.na(Sale_Price)))
ames_split <- initial_split(ames_known_price, prop = 0.9)
ames_train <- training(ames_split)
ames_test <- testing(ames_split)

ames_folds <- vfold_cv(ames_train, v = 5)

ridge_reg_spec <- linear_reg(mixture = 0, penalty = 0.1) %>%
  set_engine("glmnet")

reg_rec <- recipe(Sale_Price ~ ., data = ames_train) %>%
  step_normalize(all_numeric()) %>%
  step_impute_knn(all_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_other(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors())

ridge_reg_wf <- workflow() %>%
 add_model(ridge_reg_spec) %>%
 add_recipe(reg_rec)

ridge_reg_cv <- ridge_reg_wf %>%
  fit_resamples(ames_folds)

ridge_reg_cv %>%
  collect_metrics() %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

.metric	.estimator	mean	n	std_err	.config
rmse	standard	0.3916875	5	0.0447422	Preprocessor1_Model1
rsq	standard	0.8484653	5	0.0206100	Preprocessor1_Model1

```r
ridge_reg_cv %>%
  collect_metrics(summarize = FALSE) %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

id	.metric	.estimator	.estimate	.config
Fold1	rmse	standard	0.5189227	Preprocessor1_Model1
Fold1	rsq	standard	0.8116721	Preprocessor1_Model1
Fold2	rmse	standard	0.4013395	Preprocessor1_Model1
Fold2	rsq	standard	0.8502987	Preprocessor1_Model1
Fold3	rmse	standard	0.2862786	Preprocessor1_Model1
Fold3	rsq	standard	0.8970876	Preprocessor1_Model1
Fold4	rmse	standard	0.4538870	Preprocessor1_Model1
Fold4	rsq	standard	0.7932557	Preprocessor1_Model1
Fold5	rmse	standard	0.2980094	Preprocessor1_Model1
Fold5	rsq	standard	0.8900122	Preprocessor1_Model1

```r
ridge_reg_fit <- ridge_reg_wf %>%
  fit(ames_train)

ridge_reg_fit %>%
  tidy() %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

term	estimate	penalty
Intercept	0.1145293	0.1
Lot_Frontage	0.0090180	0.1
Lot_Area	0.0325429	0.1
Year_Built	0.0405505	0.1
Year_Remod_Add	0.0575928	0.1
Mas_Vnr_Area	0.0556967	0.1
BsmtFin_SF_1	-0.0060892	0.1
BsmtFin_SF_2	0.0097812	0.1
Bsmt_Unf_SF	-0.0252520	0.1
Total_Bsmt_SF	0.0730677	0.1
First_Flr_SF	0.0906649	0.1
Second_Flr_SF	0.1088686	0.1

Low_Qual_Fin_SF	-0.0065341	0.1
Gr_Liv_Area	0.1583716	0.1
Bsmt_Full_Bath	0.0350655	0.1
Bsmt_Half_Bath	-0.0081504	0.1
Full_Bath	0.0515726	0.1
Half_Bath	0.0267011	0.1
Bedroom_AbvGr	-0.0129840	0.1
Kitchen_AbvGr	-0.0302027	0.1
TotRms_AbvGrd	0.0400075	0.1
Fireplaces	0.0580324	0.1
Garage_Cars	0.0644812	0.1
Garage_Area	0.0316236	0.1
Wood_Deck_SF	0.0185118	0.1
Open_Porch_SF	-0.0047602	0.1
Enclosed_Porch	0.0160070	0.1
Three_season_porch	0.0060659	0.1
Screen_Porch	0.0442458	0.1
Pool_Area	-0.0145346	0.1
Misc_Val	-0.0607332	0.1
Mo_Sold	-0.0052524	0.1
Year_Sold	-0.0152585	0.1
Longitude	0.0105057	0.1
Latitude	0.0559548	0.1
MS_SubClass_One_Story_1945_and_Older	-0.0576545	0.1
MS_SubClass_One_Story_1946_and_Newer_All_Styles	0.0403500	0.1
MS_SubClass_One_Story_PUD_1946_and_Newer	-0.0429358	0.1
MS_SubClass_Two_Story_1946_and_Newer	0.0062131	0.1
MS_SubClass_other	0.0099330	0.1
MS_Zoning_Residential_Medium_Density	-0.0708960	0.1
MS_Zoning_other	-0.0910336	0.1
Street_other	-0.2907418	0.1
Alley_other	-0.0013009	0.1
Lot_Shape_Slightly_Irregular	0.0289572	0.1
Lot_Shape_other	0.0239668	0.1
Land_Contour_other	-0.0010353	0.1
Utilities_other	-0.1868721	0.1
Lot_Config_CulDSac	0.1272896	0.1
Lot_Config_Inside	0.0036636	0.1
Lot_Config_other	-0.0708138	0.1

Land_Slope_other	0.0616860	0.1
Neighborhood_Edwards	-0.1124247	0.1
Neighborhood_Gilbert	-0.2014004	0.1
Neighborhood_North_Ames	-0.0637403	0.1
Neighborhood_Northridge_Heights	0.2314445	0.1
Neighborhood_Old_Town	-0.0818374	0.1
Neighborhood_Somerset	0.1126527	0.1
Neighborhood_other	0.0598542	0.1
Condition_1_Norm	0.1254967	0.1
Condition_1_other	0.0255332	0.1
Condition_2_other	-0.0010087	0.1
Bldg_Type_TwnhsE	-0.1392049	0.1
Bldg_Type_other	-0.1470693	0.1
House_Style_One_Story	0.0360186	0.1
House_Style_Two_Story	-0.0332128	0.1
House_Style_other	-0.0116643	0.1
Overall_Qual_Average	-0.0230210	0.1
Overall_Qual_Below_Average	-0.0388567	0.1
Overall_Qual_Good	0.0166177	0.1
Overall_Qual_Very_Good	0.1862754	0.1
Overall_Qual_other	0.3649797	0.1
Overall_Cond_Average	-0.0271776	0.1
Overall_Cond_Good	0.0616089	0.1
Overall_Cond_other	-0.0459818	0.1
Roof_Style_Hip	0.0638390	0.1
Roof_Style_other	-0.0550882	0.1
Roof_Matl_other	0.0272135	0.1
Exterior_1st_MetalSd	0.0237034	0.1
Exterior_1st_Plywood	0.0056861	0.1
Exterior_1st_VinylSd	0.0098063	0.1
Exterior_1st_Wd.Sdng	-0.0006277	0.1
Exterior_1st_other	0.0772224	0.1
Exterior_2nd_MetalSd	0.0221860	0.1
Exterior_2nd_Plywood	-0.0422859	0.1
Exterior_2nd_VinylSd	0.0116312	0.1
Exterior_2nd_Wd.Sdng	0.0438734	0.1
Exterior_2nd_other	-0.0090969	0.1
Mas_Vnr_Type_None	0.0733293	0.1
Mas_Vnr_Type_Stone	0.0369354	0.1

Mas_Vnr_Type_other	-0.1086331	0.1
Exter_Qual_Typical	-0.0823973	0.1
Exter_Qual_other	0.2000401	0.1
Exter_Cond_Typical	-0.0097492	0.1
Exter_Cond_other	-0.1788661	0.1
Foundation_CBlock	-0.0268604	0.1
Foundation_PConc	0.0544823	0.1
Foundation_other	0.0036561	0.1
Bsmt_Qual_Good	-0.1680415	0.1
Bsmt_Qual_Typical	-0.1539823	0.1
Bsmt_Qual_other	-0.1530138	0.1
Bsmt_Cond_other	-0.0339487	0.1
Bsmt_Exposure_Gd	0.1728179	0.1
Bsmt_Exposure_Mn	-0.0637248	0.1
Bsmt_Exposure_No	-0.1024973	0.1
Bsmt_Exposure_other	-0.0701788	0.1
BsmtFin_Type_1_BLQ	-0.0084944	0.1
BsmtFin_Type_1_GLQ	0.0905850	0.1
BsmtFin_Type_1_LwQ	-0.0521331	0.1
BsmtFin_Type_1_Rec	-0.0100155	0.1
BsmtFin_Type_1_Unf	-0.0259222	0.1
BsmtFin_Type_1_other	-0.0398366	0.1
BsmtFin_Type_2_other	-0.0240291	0.1
Heating_other	-0.0193442	0.1
Heating_QC_Good	-0.0305130	0.1
Heating_QC_Typical	-0.0654624	0.1
Heating_QC_other	-0.1295312	0.1
Central_Air_Y	0.0546754	0.1
Electrical_SBrkr	0.0038786	0.1
Electrical_other	0.0004011	0.1
Kitchen_Qual_Good	-0.1455608	0.1
Kitchen_Qual_Typical	-0.1764373	0.1
Kitchen_Qual_other	-0.2046404	0.1
Functional_other	-0.1906057	0.1
Fireplace_Qu_No_Fireplace	-0.0047285	0.1
Fireplace_Qu_Typical	-0.0249439	0.1
Fireplace_Qu_other	0.0070166	0.1
Garage_Type_BuiltIn	0.0000858	0.1
Garage_Type_Detchd	-0.0216167	0.1
Garage_Type_No_Garage	0.0045639	0.1

Garage_Type_other	-0.1450426	0.1
Garage_Finish_No_Garage	0.0059326	0.1
Garage_Finish_RFn	-0.0492982	0.1
Garage_Finish_Unf	-0.0234584	0.1
Garage_Qual_Typical	-0.0175402	0.1
Garage_Qual_other	0.0271115	0.1
Garage_Cond_Typical	0.0293285	0.1
Garage_Cond_other	-0.0765959	0.1
Paved_Drive_Paved	0.0517364	0.1
Paved_Drive_other	0.0665841	0.1
Pool_QC_other	0.1142025	0.1
Fence_No_Fence	-0.0257223	0.1
Fence_other	-0.0121176	0.1
Misc_Feature_other	0.0751086	0.1
Sale_Type_WD	-0.0564736	0.1
Sale_Type_other	-0.0850875	0.1
Sale_Condition_Normal	0.0919020	0.1
Sale_Condition_Partial	0.1426420	0.1
Sale_Condition_other	0.0804423	0.1

From the regression output above, we can see lots of very small coefficients attached to the majority of our available predictors.

Implementing the LASSO

The work required to construct a LASSO model is nearly identical. We simply use `mixture = 1` in the model specification to signal that we want to use the LASSO constraint instead of the ridge constraint.

```r
lasso_reg_spec <- linear_reg(mixture = 1, penalty = 0.1) %>%
  set_engine("glmnet")

lasso_reg_wf <- workflow() %>%
 add_model(lasso_reg_spec) %>%
 add_recipe(reg_rec)

lasso_reg_cv <- lasso_reg_wf %>%
  fit_resamples(ames_folds)

lasso_reg_cv %>%
  collect_metrics() %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

.metric	.estimator	mean	n	std_err	.config
rmse	standard	0.4691218	5	0.0441989	Preprocessor1_Model1
rsq	standard	0.8052893	5	0.0220063	Preprocessor1_Model1

```
lasso_reg_cv %>%
  collect_metrics(summarize = FALSE) %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

id	.metric	.estimator	.estimate	.config
Fold1	rmse	standard	0.5972859	Preprocessor1_Model1
Fold1	rsq	standard	0.7851941	Preprocessor1_Model1
Fold2	rmse	standard	0.5053414	Preprocessor1_Model1
Fold2	rsq	standard	0.7846345	Preprocessor1_Model1
Fold3	rmse	standard	0.3694636	Preprocessor1_Model1
Fold3	rsq	standard	0.8542007	Preprocessor1_Model1
Fold4	rmse	standard	0.5049735	Preprocessor1_Model1
Fold4	rsq	standard	0.7445101	Preprocessor1_Model1
Fold5	rmse	standard	0.3685445	Preprocessor1_Model1
Fold5	rsq	standard	0.8579072	Preprocessor1_Model1

```
lasso_reg_fit <- lasso_reg_wf %>%
  fit(ames_train)

lasso_reg_fit %>%
  tidy() %>%
  filter(estimate != 0) %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

term	estimate	penalty
Intercept	0.0226717	0.1

Year_Built	0.0942332	0.1
Year_Remod_Add	0.0680334	0.1
Mas_Vnr_Area	0.0352078	0.1
Total_Bsmt_SF	0.1427965	0.1
Gr_Liv_Area	0.3128844	0.1
Fireplaces	0.0542894	0.1
Garage_Cars	0.0870143	0.1
Garage_Area	0.0452995	0.1
Neighborhood_Northridge_Heights	0.2164590	0.1
Bldg_Type_other	-0.0043000	0.1
Overall_Qual_Very_Good	0.0590885	0.1
Overall_Qual_other	0.3117465	0.1
Exter_Qual_Typical	-0.1080192	0.1
Exter_Qual_other	0.0160573	0.1
Bsmt_Exposure_Gd	0.1342189	0.1
BsmtFin_Type_1_GLQ	0.0823411	0.1
Kitchen_Qual_Typical	-0.0661239	0.1

We can see in the LASSO model that many of the predictors were assigned coefficients of 0. The LASSO procedure identified that those predictors weren't worth "buying", so it left them out and only included the most useful predictors!

Summary

In this notebook, we introduced two new regression models: *Ridge Regression* and the *LASSO*. These models are of the familiar form $\mathbb{E}\left[y\right] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$, but are fit using a *constrained* optimization procedure rather than *ordinary least squares*. When we implement Ridge and the LASSO, we are reducing the likelihood that the resulting model is overfit. With these models, however, we must remember that all numerical predictors need to be *scaled* and no missing data can be present in the training data.