Problem Set #3

Kevin McAlister

February 2nd, 2022

This is the third problem set for QTM 385 - Intro to Statistical Learning. This homework will cover applied exercises related to predictor selection for linear regression models.

Please use the intro to RMarkdown posted in the Intro module and my .Rmd file as a guide for writing up your answers. You can use any language you want, but I think that a number of the computational problems are easier in R. Please post any questions about the content of this problem set or RMarkdown questions to the corresponding discussion board.

Your final deliverable should be two files: 1) a .Rmd/.ipynb file and 2) either a rendered HTML file or a PDF. Students can complete this assignment in groups of up to 3. Please identify your collaborators at the top of your document. All students should turn in a copy of the solutions, but your solutions can be identical to those of your collaborators.

This assignment is due by February 11th, 2022 at 11:59 PM EST.

Problem 1: Some Cool Ridge and LASSO Identities (30 pts.)

Part 1 (10 pts.)

Both ridge regression and the LASSO are approaches that add penalty terms to the standard mean squared error loss function that discourage solutions that are too dense - we'd rather have a biased estimate of the regression coefficients that then leads to lower expected prediction error by virtue of reduced between model variance. Both the ridge and LASSO solutions arise due to almost ad-hoc solutions to the problem of collinearity and/or pursuit of a method for solving the L_0 best-subset regression problem. However, there is an equally useful equivalent justification via Bayesian statistics.

Recall that the likelihood for the general linear regression problem assumes normal idiosyncratic errors:

$$\ell(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\beta}, \sigma^2) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(y_i - \boldsymbol{x}_i'\boldsymbol{\beta})^2\right]$$

and admits a workable log-likelihood:

$$\ell\ell(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\beta}, \sigma^2) = -\frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$$

The Bayesian approach places a **prior** on the unknown parameters and characterizes a **posterior distribution** that combines our prior uncertainty about the value of the unknown parameters with uncertainty due to sampling error (e.g. the prior and the likelihood).

Bayes' theorem tells us that the log posterior distribution on $\boldsymbol{\beta}$ taking σ^2 as known is proportional to ($\propto$):

$$\log f(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y}, \sigma^2) \propto \ell\ell(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\beta}, \sigma^2) + \log f(\boldsymbol{\beta})$$

where $\log f(\boldsymbol{\beta})$ is the **log prior**. Then, the point estimate of $\boldsymbol{\beta}$ that minimizes the loss is the one that **maximizes** the posterior - the posterior mode.

Suppose we define the prior over $\boldsymbol{\beta}$ as the product of independent normal distributions with common mean 0 and variance τ^2:

$$f(\boldsymbol{\beta}) = \prod_{j=1}^{P} \mathcal{N}(\beta_j \mid 0, \tau^2) = \prod_{j=1}^{P} \frac{1}{\sqrt{2\pi\tau^2}} \exp\left[-\frac{1}{2\tau^2}\beta_j^2\right]$$

Show that the negative log posterior distribution on $\boldsymbol{\beta}$ is proportional to the ridge regression loss function:

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \lambda \sum_{j=1}^{P} \beta_j^2$$

and find an expression for the regularization constant, λ that is a function of known quantities.

Notes:

1. σ^2 is the same error variance we've seen before. We can think of τ^2 as an information penalty - compared to OLS, I need to see $\frac{1}{\tau^2}$ more evidence to confirm that I am better off **not** setting $\beta_j = 0$.
2. $\boldsymbol{\beta}$ is the random variable here. Under the Bayesian paradigm, we think of parameters as uncertain. So, try to construct everything keeping $\boldsymbol{\beta}$ in mind.
3. Every step of the way, you can get rid of terms that don't have anything to do with $\boldsymbol{\beta}$ - since we're finding an equation that is proportional to the true posterior, we can get rid of products (or sums in log space) that don't rely on the random variable. The leftovers can all be thrown to the unimportant (for this purpose) **normalizing constant** at the front of proper PDFs that make everything integrate to one.

Part 2 (10 pts.)

There's nothing saying that our prior on each β_j needs to be normal. In fact, we can choose other priors for a variety of different reasons. Another viable choice of prior on β_j is the Laplace distribution with mean 0 and scale parameter $b > 0$:

$$f(\boldsymbol{\beta}) = \prod_{j=1}^{P} \frac{1}{2b} \exp\left[-\frac{|\beta_j|}{b}\right]$$

The variance of a Laplace distributed random variable is $2b^2$ - so it scales similarly to variance in the normal distribution.

Show that the negative log posterior distribution on $\boldsymbol{\beta}$ using Laplace priors is proportional to the LASSO loss function:

$$\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \lambda \sum_{j=1}^{P} |\beta_j|$$

and find an expression for the regularization constant, λ that is a function of known quantities.

Part 3 (10 pts.)

There is a relationship between the optimal solutions for the regression coefficients under no penalty (the OLS solution), the ridge penalty, and the LASSO penalty. The exact relationship cannot be derived for most cases, but we can gain some knowledge by assuming that the predictors are exactly orthogonal to one another. Since we can always rescale the variance of the features, we can further restrict this to feature sets that have **orthonormal** columns - $\boldsymbol{X'X} = \mathcal{I}_P$.

Assuming the feature matrix, $\boldsymbol{X}$, has orthnormal columns, show that:

$$\hat{\beta}_{OLS} = \boldsymbol{X'y} \; ; \; \hat{\beta}_{Ridge} = \frac{\hat{\beta}_{OLS}}{1+\lambda} \; ; \; \hat{\beta}_{LASSO} = \text{sign}(\hat{\beta}_{OLS}) \times \max(|\hat{\beta}_{OLS}| - \lambda, 0)$$

What do these equations show about **how** ridge and LASSO shrink coefficients towards zero? Why can the LASSO set a coefficient to zero while ridge cannot?

Notes:

1. $\sum_{j=1}^{P} \beta_j^2$ can also be expressed as $\beta'\beta$.
2. Expand the squared-L2 norm first and then substitute $\hat{\beta}_{OLS} = \boldsymbol{X'y}$.
3. It is probably easier to drop everything to elements of the coefficient vector for the LASSO at a certain point - e.g. work with β_j instead of $\boldsymbol{\beta}$. Since everything becomes a sum or sum of squares or sum of absolute values, you can separate the problem out easier this way.
4. You can take it as a given that the sign of $\hat{\beta}_{OLS,j}$ is the same as $\hat{\beta}_{LASSO,j}$ (unless the LASSO solution is zero, but it won't matter there).
5. These solutions can be found online and in various textbook resources. I think this is a great exercise for thinking through complex minimization problems, so don't spoil yourself unless you really find yourself stuck.

Problem 2: Applying Ridge and LASSO (70 pts.)

Ridge and LASSO are great methods for parsing through data sets with lots of predictors to find:

1. An interpretable set of important predictors - which predictors are **signal** and which ones are just **noise**
2. The set of parameters that minimize expected prediction error (with all the caveats that we discussed in the previous lectures)

Where these methods really shine for purpose 1 (and purpose 2, by construction) is when the ratio of predictors to observations approaches 1. To see this and work through an example using pre-built software, let's try to build a model that predicts IMDB ratings for episodes of the Office (the U.S. Version). `office_train.csv` includes IMDB ratings (`imdb_rating`) for 115 episodes of the office and a number of predictors for each episode:

1. The season of the episode (1 - 9, which should be treated as an unordered categorical variable)
2. The number of times main characters speak in the episode (`andy` through `jan`)
3. The director of the episode (`ken_kwapis` through `justin_spitzer`). There can be more than 1 director per episode, so it's not a pure categorical variable. However, the correlation is high!

Let's use this data to build a predictive model for IMDB ratings and check our predictive accuracy on the heldout test set (`office_test.csv`).

For this problem, you can restrict your search to the set of standard linear models (e.g. no interactions, no basis expansions, etc.). If you would like to try to include more terms to improve the model, you are more than welcome to try!

Part 1 (10 pts.)

Start by limiting yourself to the standard OLS model.

Find the regression coefficients that minimize the training error under squared error loss and use this model to compute the LOOCV estimate of the expected prediction error.

Which predictors are important? Which ones are not? This can be difficult to tell from the OLS estimates!

Part 2 (20 pts.)

Now, consider ridge regression. Using a pre-built implementation of ridge regression, train the model using a large number of possible values for λ.

For each value of λ used, compute the L1-norm for the estimated coefficients (e.g. $\sum |\beta_j|$) and plot the value of the regression coefficients against this value - there should be a separate line for each regression coefficient. (Hint: There is a built-in method for doing this in the `glmnet` package.) Which predictors seem to be most important? You can see these as the one with "non-zero" regression coefficients when λ is large or the L2-norm for the estimated coefficient set is small. If it is too difficult to see over the entire λ path, restrict the x variable limits to the lower part of the graph with the `xlim = c(low,high)` argument. It may still be kind of difficult to tell from the graph - ridge regression is not known for its pretty pictures!

Finally, we need to select a value of λ that minimizes the expected prediction error. Using 10-fold cross validation, find a reasonable value of λ that should minimize the expected prediction error. You can choose the actual minimum or a slightly less complex model (smaller λ is less complex). Defend this choice.

Create a plot that demonstrates the regression coefficients for the ridge regression with your optimal choice of λ. Which predictors are important? Which ones are not? I recommend using a sideways bar plot - you can see an example construction here.

Part 3 (20 pts.)

Finally, consider linear regression with the LASSO penalty. Using a pre-built implementation, train the model using a large number of possible values for λ.

For each value of λ used, compute the L1-norm for the estimated coefficients (e.g. $\sum |\beta_j|$) and plot the value of the regression coefficients against this value - there should be a separate line for each regression coefficient. Which predictors seem to be most important? You can see these as the one with non-zero regression coefficients when λ is large or the L1-norm for the estimated coefficient set is small.

Finally, we need to select a value of λ that minimizes the expected prediction error. Using 10-fold cross validation, find a reasonable value of λ that should minimize the expected prediction error. You can choose the actual minimum or a slightly less complex model (smaller λ is less complex). Defend this choice.

Create a plot that demonstrates the regression coefficients for the LASSO regression with your optimal choice of λ. Which predictors are important? Which ones are not?

Part 4 (20 pts.)

Which of OLS, Ridge, or LASSO has the smallest cross validation estimate of expected prediction error? Do you have any intuition as to why this result occurs?

Using the optimal models from each step, compute an estimate of the expected prediction error using the heldout test data. Does the same relationship hold?

Create a plot (or set of plots) that puts the predicted test set outcome for each method along the x-axis and the true value on the y-axis. How does OLS compare to Ridge and LASSO? How do the regularized models improve the predictive fit?

Do any of the models provide what you might consider to be a "good" predictive model? Interpretable?

Discussion on Computational Resources

I'm going to provide some guidance on implementing Ridge and LASSO in R and Python. For these problems, I highly recommend using the package `glmnet` in both R and Python. The vignette for R can be found here and the vignette for the Python package can be found here. This package was developed by the authors of ESL and ISLR as a general purpose implementation of the Elastic Net for estimating regression coefficients. The packages is C++ and Fortran optimized, meaning that it absolutely beats the pants off of other implementations in terms of computational speed. It also has built in functions and favorable construction for plotting regularization paths and choosing λ via cross validation. There are a million and one more implementations, but I think that this one is the best for our purposes.

A first note is that the main function `glmnet` does not take formula inputs - e.g. `y ~ x1 + x2`. Rather, it requires that you pass it a vector or matrix of outcome values, `y`, and a $N \times P$ matrix of predictors, `x`. This is a design choice that is implemented to prevent people from using `glmnet` to mine their way to "victory" in finding the best predictive model. One thing that we've seen is that we may want to consider basis expansions (`poly()` in R) or multiplicative interactions between predictors (`x1*x2` in R). The formula language makes this implementation easy - maybe too easy. Since Ridge and LASSO are essentially indiscriminately setting some number of predictors to zero, this would lead to nonsense models that have interaction terms without their constituent marginal components and broken basis expansions. Rather than implement smart grouping features, the authors of the package discourage this by making you construct the predictor set rather than creating "user-friendly" niceties - when you see how big P is for the full model, you'll often rethink whether or not you truly believe it's necessary!

`glmnet` models the intercept separately. This is because it makes little since to penalize the estimate of the intercept. So, you do not need to include a column of 1s in your predictor matrix.

There is one case where this construction leads to some pain - dummying out or one-hot encoding unordered categorical predictors. For example, the season in the Office problem above should be treated as an unordered categorical predictor. Typically, we can set `season` to be a factor and the formula language in R will deal with it for you. However, `glmnet` requires us to do it ourselves.

To process the training and test data to "one-hot encode" the season of the episode, we need to convert the unordered categorical variable with M categories to M separate predictors that take a value of 1 if they are in that group and 0 otherwise. For example, a variable called `Season 9` would take a value of 1 if the episode was in season 9 and 0 otherwise. If the unordered categorical variable is a factor in a data frame, then there is an easy way to do it in R using the `caret` package and the function `dummyVars`:

```
train$season <- factor(train$season)
dummy_train <- caret::dummyVars(" ~ .", data = train)
train <- data.frame(predict(dummy_train, newdata = train))
```

which takes a data frame with factor columns and returns another data frame with all factor columns dummied out and the rest of the columns intact. In Python, `OneHotEncoder` in `sklearn` will achieve the same thing. There are also ways to do this manually, so do what is easiest for you.

Another point is that `glmnet` standardizes all of the predictors. Using a single λ shrinkage parameter works best when all predictors are on the same scale. This leads to standardized regression coefficients that are interpreted similarly - a 1 standard deviation change in X results in a β unit change in Y. This standardization can be done by the user, but it is not recommended since `glmnet` implements the standardization in a specific way. This can lead to some weirdness when trying to predict new values of the outcome using a specific λ value, so it is recommended that you use the built-in `predict` function to do this.

`glmnet` has a lot of arguments. But, we only really need to look at a few of them:

1. `x` and `y` - the input predictors and outcomes
2. `family` - for now, we only need `family = "gaussian"`. `glmnet` can do ridge or LASSO penalized generalized linear models - most importantly logistic regression. Since we're only working with linear regression, we are assuming Gaussian distributed errors.
3. `alpha` - the value for the exponent of the Elastic Net penalty. `alpha = 0` corresponds to ridge regression while `alpha = 1` corresponds to LASSO regression.
4. `nlambda` - the number of λ values to consider when fitting the regularization path. `glmnet` is smart and figures out good values for the minimum and maximum λ to be considered. This controls how granular the path of λ checked is. The default of 100 is usually more than enough.
5. `lambda` - there's no need to do this! Just fit the full path of λ values and extract via `predict` or `coefs`.

Everything else is only needed for very specific circumstances. If you need to change one of these from the defaults, you'll know.

Two other functions of importance: 1) `predict` and 2) `cv.glmnet`.

`predict` for a `glmnet` object will take in new values of x and return the value of $\hat{y}$. `newx` should be a `M \times P` matrix of M new observations. `s` is the value of λ at which predictions are to be made. `type` is not relevant for linear regression, but can be used to change what level of $\hat{y}$ is computed (more to come in a few weeks).

`cv.glmnet` will create random partitions within the provided training data to do K-fold cross validation. This approach is 100% superior to any implementation you will write yourself, so just use this for computing cross-validation estimates of the expected prediction error. The arguments for `cv.glmnet` include all of the arguments for `glmnet` with two additional important inputs:

1. `nfolds` - the number of disjoint folds to create within the training data. Default is 10 and is usually a good choice.
2. `type.measure` - the loss function to consider when computing the estimate of the expected prediction error. The default is to compute deviance, $-2\ell\ell$, for the regression model. Note that this is equivalent to MSE for the Gaussian model. Two other common choices - `mse` and `mae`. The choice here should be informed by what loss you're trying to minimize. MSE works well for our purposes, but may not make sense in other contexts.

With `cv.glmnet`, you can plot the K-fold curve for all attempted values of λ and search for a meaningful minimum. It will also return the minimum loss λ and the one standard error value of λ. Your choice here should be informed by the problem, but is unlikely to make a significant difference in your conclusions.