Regularized Linear Regression

Exploring regularization with simulated data

A note on `glmnet`

Here, I'll cover two important points about the behavior of the `glmnet` package.

Passing in data

For `lm()`, you passed in the entire data frame, including both target variable and predictors. `glmnet(features, target, ...)` andcv.glmnet(features, target, ...)`expect a *scaled matrix of predictors* for`features`and a numeric vector for`target. The `scale()` function returns a matrix, so you can just call `scale()` on a data frame of predictors and pass that in as `features`.

Picking values of λ

"Ordinarily", one might expect that, for every different value of λ we want to try using with regularized linear regression, we would have to recompute the entire model from scratch. However, the `glmnet` package, through which we'll be using regularized linear regression, will automatically compute the regression coefficients for *a wide range of λ values$ simultaneously:

> The `glmnet` algorithms use cyclical coordinate descent, which successively optimizes the objective function over each parameter with others fixed, and cycles repeatedly until convergence. The package also makes use of the strong rules for efficient restriction of the active set. Due to highly efficient updates and techniques such as warm starts and active-set convergence, our algorithms can compute the solution path very fast.

When you call `glmnet()` – or, later, `cv.glmnet()` – you'll get out an object, `fit`. (You should generally not be specifying *which* λ values the algorithm should use at this point – it'll try to determine that on its own.) By printing out `fit` in the console, you can see which values of λ were used by `glmnet`.

When you want to make predictions with this `fit` object, you'll have to specify *which* value of λ to use – instead of calling `predict(fit, new_data)`, you'll want to call `predict(fit, new_data, s=lambda)` for some particular λ = `lambda`. Similarly, when extracting coefficients, you'll want to call `coef(fit, s=lambda)`.

Finally, `cv.glmnet()` will use *cross-validation* to determine `fit$lambda.min` and `fit$lambda.1se`. The former is the value of λ (out of all those the algorithm evaluated) which minimizes the cross-validated mean squared error (MSE), and the latter is the greatest value of λ (again, of those evaluated by `glmnet`) such that the MSE corresponding to `fit$lambda.1se` is within 1 standard error of the MSE corresponding to `fit$lambda.min`.

If it turns out that the optimal value of λ lies at either end of the range of λ values used by `glmnet`, then you'll want to modify the range of λ. However, never pass in just a single value for the `lambda` parameter of `glmnet()` and `cv.glmnet()`, instead modifying `nlambda` and `lambda.min.ratio`:

> Typical usage is to have the program compute its own `lambda` sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of `lambda` overrides this. WARNING: use with care. Do not supply a single value for `lambda` (for predictions after CV use `predict()` instead). Supply instead a decreasing sequence of `lambda` values. `glmnet` relies on its warms starts for speed, and it's often faster to fit a whole path than compute a single fit.

Comparing regularization and stepwise regression

We'll continue using the simplified speed dating dataset from yesterday. You can restrict to a particular gender or use the whole dataset as you prefer.

Using the entire dataset

The `glmnet()` and `cv.glmnet()` functions can perform both L^1 and L^2 regularized linear regression as well as a mix of the two (which we'll be exploring later). This behavior can be tuned via the `alpha` parameter; read the official documentation to figure out it works.

- Pick one of the five rating variables and use backward stepwise regression to generate predictions for the whole dataset. (Don't use cross-validation at this point.)

- Use `glmnet()` to generate similar predictions with both L^1 and L^2 regularized linear regression.
  - Look at which values of λ were used by `glmnet()`.
  - Write a function that (1) takes the model object generated by a call to `glmnet()` and the true values for the target variable, (2) uses `predict()` to generate predictions for every value of λ which `glmnet()` tried, and (3) returns the λ corresponding to the lowest RMSE and the RMSE itself.
- Compare the minimum RMSE for both regularized fits with the RMSE for backward stepwise regression.
- Compare and interpret the coefficients for L^1 and L^2 regularized linear regression using the optimal values of λ determined earlier.

Making cross-validated RMSE predictions

As you saw in the assignment on resampling, we want to use *cross-validation* to get more accurate estimates of model quality. In particular, stepwise regression tends to *overfit*, because of problems with multiple hypothesis testing, so non-cross-validated estimates of a stepwise regression model's quality are often overly optimistic. (However, it's easy to understand and, pedagogically, a good stepping stone to regularization, which is why we include it in our curriculum.)

- Briefly skim Stopping stepwise: Why stepwise and similar selection methods are bad ... for an introduction to some of the problems with stepwise linear regression.

Write a function following these specifications:

- Take as input one of the five ratings (`"attr_o"`, `"intel_o"`, ...), for which you will generate predictions. Also, take as input a `gender` parameter and filter the data for the selected gender.
- Use 10-fold cross validation to generate predictions for the selected rating with stepwise regression, L^1 regularized linear regression, and L^2 regularized linear regression.
- For regularized linear regression, use `cv.glmnet()` to get cross-validated estimates of the optimal value of λ. As such, when generating predictions for an regularized linear model `fit`, use the value of λ stored in `fit$lambda.min`.
- Return the RMSE associated with each of the three sets of predictions.

Here are some points to keep in mind:

- Within each cross-validation fold, you'll want to `scale()` the features which you pass into `cv.glmnet()`. When generating predictions on the *held-out* data, you want to scale the features in the same way (*i.e.*, by applying the same linear transformation). The output of `scale()` will contain *attributes* which can be accessed and passed into successive calls of `scale()` to perform the same transformation.

- If you have a string, say, `"attr_o"`, and you want to pass that into `lm()` as part of the regression formula, you can paste together the formula's components (*e.g.*, `paste("attr_o", "~.")`), call `formula()` on the string to turn it into a *formula*, and then passing the formula into `lm()`.

Use your function to explore the difference in model quality between backward stepwise regression, L^1 regularized regression, and L^2 regularized regression when predicting each of the five different ratings.

Elastic net regression

Instead of penalizing the sum of squared residuals by the L^1 or L^2 norm of the regression coefficients, we can penalize with a combination of the two, corresponding to setting the `alpha` parameter in `glmnet()` to a value between 0 and 1. We can use cross-validation to find the optimal *pair* of *hyperparameters* (α, λ).

Thankfully, we won't have to implement that ourselves (for now)! Instead, we can use the `caret` package to get a cross-validated estimate of the optimal (α, λ).

Here's an example of how to use the `caret` package's `train()` function:

```
param_grid = expand.grid(.alpha = 1:10 * 0.1, .lambda = 10^seq(-3, 0, length.out=10))
control = trainControl(method="repeatedcv", number=10, repeats=3, verboseIter=TRUE)
caret_fit = train(x=features, y=target, method="glmnet", tuneGrid=param_grid, trControl=c
```

In the above example, we perform *10-fold cross-validation* repeated *3 times*.

Whereas `cv.glmnet()` picks an appropriate sequence of λ values, the `caret` package does not.

- Run `cv.glmnet()` on the data for any rating to get a rough sense for what the range of λ should be, and then use that when doing grid search with `caret`.

Write a function according to the following specifications:

- Take as input one of the five rating variables to make predictions for. Also, take as input a gender parameter to filter by.
- Use the `caret` package, following the above example, to find the optimal values for (α, λ).
- Calculate the corresponding RMSE and compare the different RMSEs for all combinations of (gender, rating).