Applying Risk Parity to Stacked Regressions in Supervised Learning

Caiyi Zhang, Chenkuan Liu

May 28, 2018

1. Introduction

This project mainly addresses two questions. First, we examine and compare the methods of model stacking and risk-parity in improving prediction accuracy. Second, we aim to find the optimal weight of each model used in both methods.

Initially proposed by Wolpert (1992), stacking is the idea that a linear combination of weak models will generally produce a better estimate in prediction. On the other hand, risk-parity in investment is to have each asset contributes equal risk to the portfolio. The data set being used is the Boston Housing data from the R package *mlbench*. The data set contains 506 observations on 14 variables, where the target variable is the median housing price. Here's a look at the data set:

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
0.02985	0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7

Clearly, the data we are dealing with is a regression problem, so we will use the root mean squared error (RMSE) as our metric, and the method with a lower RMSE will be the better one.

2. Training Models with 10-Fold CV

Randomly partition the Boston Housing data set into 5 folds. In each single process, we train the models based on four folds (this will be the training set) and use the remaining one (call it the test set) for testing. Proceed for five times so that each fold is used as a test set. This is a necessary means to prevent biases and overfitting.

Similarly, within each training set, we resample the data by 10-fold cross validation. The purpose is to evaluate the model performance on the out-of-sample data, namely the test sample. 10-fold cross validation will further split the training set into 10 subsets (randomly), among these one will be kept as the validation set for testing the model, the remaining 9 subsets are used for actual training. The process will repeat 10 times so that each of the ten subsamples are used exactly once as a validation set; it yields 10 results for RMSE, which will then be averaged into a single training error. According to Breiman's *Stacked Regressions*, 10-fold CV is both more computationally efficient and more accurate than leave-one-out cross validation (62).

In each round, we use five base models to train the data by using functions from the *caret* and *glmnet* package. Specifically, the models are linear regression (LM), ridge regression, lasso regression, classification and regression trees (CART), and support vector machines (SVM). In training LM, CART, and SVM, we preprocess the data by centering and scaling

(for ridge and lasso, the function `cv.glmnet` normalizes the data automatically) to ensure units of regression coefficients are on the same scale.

3. Stacking with Gradient Descent

Let X_i represent the i^{th} model used in this experiment $i = 1, 2, ..., 5$. Take a linear combination of the selected models

$$\hat{Y} = \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4 + \alpha_5 X_5$$

then $\hat{Y}$ represents the final predicted value, and the coefficient α_i is the weight of each model.

To simplify the above, we turn it into $\hat{Y} = X\alpha$. X is a $m \times 5$ matrix where m equals the number of cases in the training set, and the each column corresponds to Model X_i's prediction on that training set. α is the weight vector with dimension 5×1, and the product $\hat{Y}$ is a $m \times 1$ vector. Typically, there are no constraints on the coefficients α_i; however, the problem being dealt with involves model stacking, and it does not make sense for α_i to have negative value. Although this is the case, we did not impose any constraint in our code as the selected models did not produce negative coefficients.

The objective is to find the optimal weights α_i such that the prediction error is minimized. That is, $\hat{Y}$ is as close to Y as possible. Define the cost function:

$$J(\alpha) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{Y}(X^{(i)}) - Y^{(i)})^2$$

Note i is each row of observation, and m is the number of observations in the training sample ($m \approx 404$). The $2m$ is just to make the derivative of $J(\alpha)$ look nice.

We want to find the weight vector α by minimizing the cost function. In this project, the gradient descent algorithm is used to solve this problem. The algorithm has the expression

$$\alpha_{i+1} = \alpha_i - \gamma J'(\alpha_i)$$

Here γ is the learning rate. The algorithm will be run a number of times, and γ is the step size we take at each iteration. Since the second term is subtracted from α_i, this will eventually lead to a movement towards the minimum, provided that γ is small enough and the number of iterations are sufficient.

Calculating the derivative $J'(\alpha_i)$ requires a bit more work since it involves vector calculus. We refer to the lecture notes on April 23rd and come to the result

$$\frac{dJ}{d\alpha} = \frac{1}{m}(X^t X\alpha - X^t Y)$$

Again, X is a $m \times 5$ matrix, α is a 5×1 column vector, Y is a $m \times 1$ column vector. So $X^t X\alpha - X^t Y$ should be a 5×1 column vector, which makes sense because each element in α corresponds to a model's weight.

Next, we apply the gradient descent function on the training sample. The function will return both the cost history and the calculated weights. To see how the weighted models

perform on the test set, we use the linear function $\hat{Y} = X\alpha$ again and assign the calculated weights to α. In addition, X_i becomes the prediction made by each model on the test set, and Y in the cost function is the original \$medv of the test set.

Finally, take the square root of the cost function (applied on the test set) gives the RMSE of the stacked models. Repeat the process for five times so that each of the five sets is tested. Average the five RMSE and the result will indicate if stacking has improved the prediction accuracy.

4. Finding Weights under Risk-parity

The idea of risk-parity can be applied to stacking models as follows: we consider different base models as "individual asset" in the portfolio, and the variance of each model represents risk. The objective is to find the models' weights such that each model contributes equally to the variance of the prediction. Expressed mathematically, each model has the same contribution λ where

$$\lambda = \frac{w' \sum w}{n}$$

In the above expression, w is the weight vector, and $w' \sum w$ is the portfolio's volatility (a value). In this case, the numerator is just the standard deviation of the final prediction. n equals the number of "assets," that is, the base models. (Li et al.) Similar to stacking with gradient descent, the weights are found using the training set and the process is run on each fold. For ease of comparison, we use the same cost function to calculate RMSE. The code implementation involves Newton's method and is based on the solution described in "Efficient Algorithms for Computing Risk Parity Portfolio Weights." (Li et al.)

5. Results

In the first part of the stacking process, we have a linear function from which we find the coefficient before each predictor variable. To solve for these coefficients, we need a cost function and an optimization algorithm (gradient descent is used in this project). From the cost function we can compute the RMSE from stacking. That is, solve for coefficients in the linear function

$$\hat{Y} = \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4 + \alpha_5 X_5$$

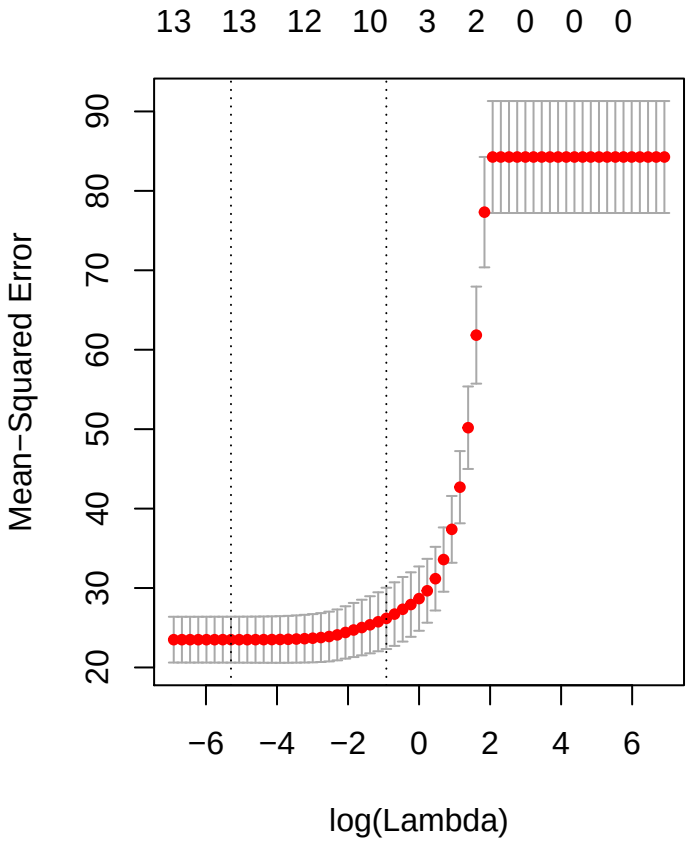
In the second part, we employ a similar idea to risk-parity except we impose an initial constraint that the coefficients should sum to 1 and that α_i is nonnegative (i.e. we consider the long option only). By definition, risk-parity requires each term $\alpha_1 X_1$, $\alpha_2 X_2$,..., $\alpha_5 X_5$ be roughly the same.

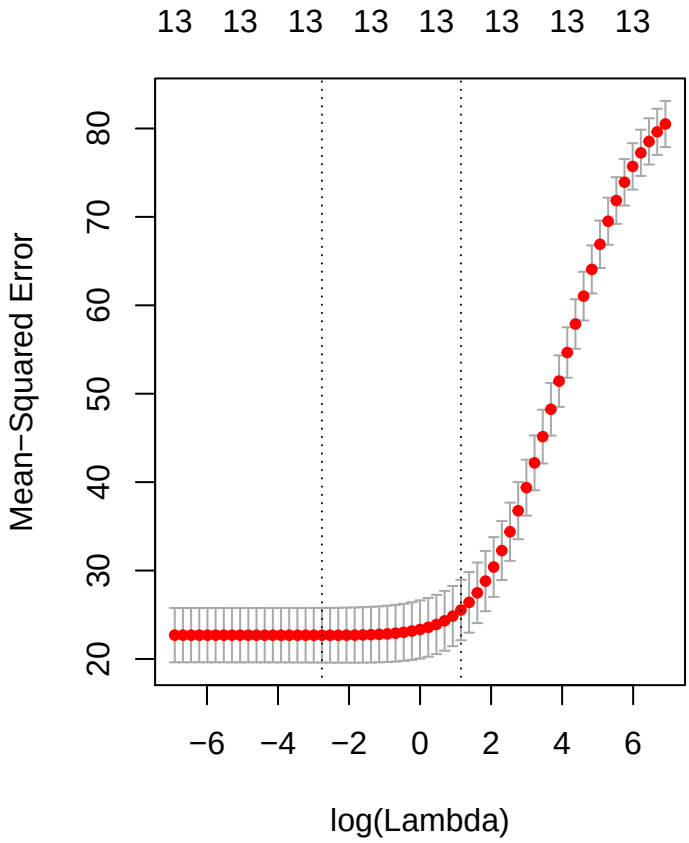
In each round two RMSEs are generated–one for stacking and one for risk-parity. Since there are five rounds in total, we can get a group of five RMSEs for stacking and another group of five RMSEs for risk-parity. Compute the average of each group, and the resulting two RMSEs would be the final measure.

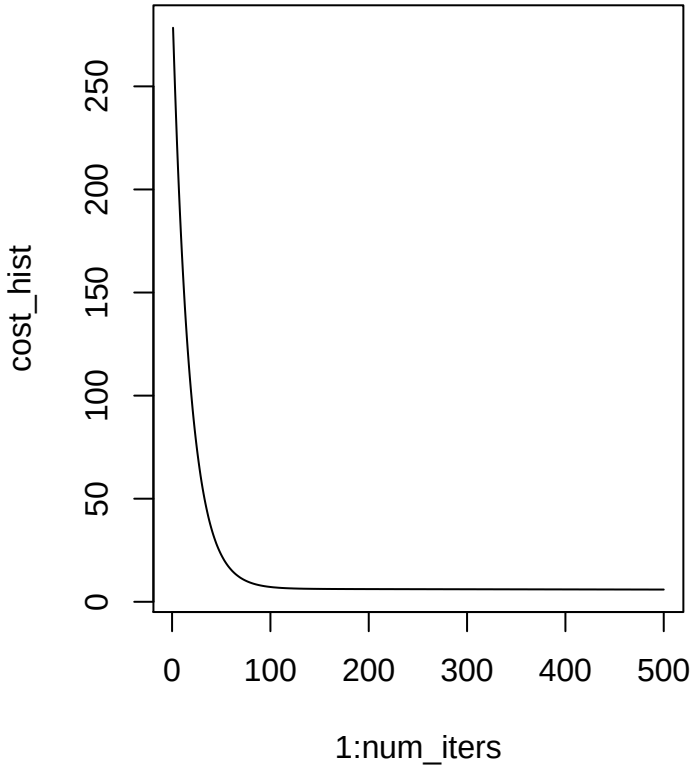
Recall earlier we split the Boston Housing data into 5 folds. We train the models on four folds and use the remaining set for testing. The process will repeat 5 times until each fold

has been used as the test set. For illustration purpose, we display plots and graphs for Round #1 only and the change in relevant numbers for all runs.

Round 1




13 13 13 13 13 13 13 13
Mean−Squared Error
20 30 40 50 60 70 80
−6 −4 −2 0 2 4 6
log(Lambda)


cost_hist
0 50 100 150 200 250
0 100 200 300 400 500
1:num_iters

Info	Stacking	Risk Parity
lm	0.1868268	0.1494703
svm	0.2183417	0.2481197
cart	0.2324934	0.3041181
lasso	0.1867259	0.1492838
ridge	0.1865494	0.1490080
Total W.	1.0109372	1.0000000
RMSE	3.0101824	2.9174017

Round 2

Info	Stacking	Risk Parity
lm	0.1851620	0.1484133
svm	0.2233768	0.2623346
cart	0.2338087	0.2938016
lasso	0.1849664	0.1480745
ridge	0.1844999	0.1473759
Total W.	1.0118139	1.0000000
RMSE	2.3880380	2.2750711

Round 3

Info	Stacking	Risk Parity
lm	0.1877359	0.1543849
svm	0.2188454	0.2509532
cart	0.2314427	0.2878607
lasso	0.1870602	0.1532530
ridge	0.1872263	0.1535482
Total W.	1.0123104	1.0000000
RMSE	2.5575596	2.4665501

Round 4

Info	Stacking	Risk Parity
lm	0.1876835	0.1468534
svm	0.2216108	0.2764953
cart	0.2271349	0.2850255
lasso	0.1872326	0.1460258
ridge	0.1869530	0.1456000
Total W.	1.0106147	1.0000000
RMSE	3.7356190	3.7126396

Round 5

Info	Stacking	Risk Parity
lm	0.1866694	0.1539205
svm	0.2241729	0.2704155
cart	0.2291157	0.2695490
lasso	0.1862065	0.1531944
ridge	0.1860031	0.1529207
Total W.	1.0121676	1.0000000
RMSE	2.4378374	2.3690592

The average of RMSE errors from these five rounds can serve as a good indicator of the model's performance on an independent data set. When `num_iters = 500`, we can see that risk parity performs better than staking.

Info	Mean RMSE	(max) Iteration
Stacking	2.825847	500
Risk Parity	2.748144	6

6. Conclusion

The above shows that stacking performs not as well as risk-parity at 500 iterations. The latter results in a smaller error. However, as the number of iterations increases, the gradient descent algorithm runs more times than before, achieving a better accuracy than risk-parity. We experiment with different iterations to see the change in RMSE (the average of five runs). Here we omit the rerunning process. After several trials, we get the values of the average RMSE with the number of iterations set to 1000, 1500, 2000, and 2500.

Risk-parity is much more efficient: it converges only after several runs, which means a rather accurate solution is found in significantly fewer steps. In fact, our experiment shows the maximum iteration is around 5 or 6. Although the prediction error from stacking is lower at `num_iters > 1500`, it does so at the expense of computational efficiency.

Compared to the thousands of iterations needed under stacking, we draw the conclusion that risk-parity is much more efficient in producing a rather accurate estimate. Therefore, risk-parity generally gives a better model performance than stacking.

iterations	stackMeanRMSE	rpMeanRMSE
500	2.825847	2.748144
1000	2.786127	2.748144
1500	2.753257	2.748144
2000	2.726300	2.748144
2500	2.704414	2.748144

7. Remarks

As shown above, the number of iteration is 1500, the RMSE of risk parity is smaller than that of stacking by 0.005. And when the number of iteration reaches 2000 or even higher, stacking becomes better than risk-parity. Since the latter converges in a few steps, change in iterations will not further reduce RMSE.

Even though a better result is achieved through stacking by simply raising the number of iterations, the running time would also increase accordingly. Moreover, from the `cost_hist` graph, we observe that when the number of iteration is above 200, the function almost looks like a horizontal line, which means a significant amount of iterations is needed to lower a tiny bit of the stacking RMSE.

In sum, whether stacking or risk-parity wins in terms of prediction accuracy depends on the iterations of stacking. Although at `num_iters=2500` or a more extreme value will continue lowering the RMSE value, it can be time-consuming, especially when the data set is large. As for risk-parity, a fairly accurate solution is found in less than 10 iterations.

8. References

Breiman, L. (1996). *Stacked Regressions*. Machine Learning, 24(1), 49-64. doi:10.1007/bf00117832

Ng, A. (n.d.). *CS229Lecturenotes - Machine learning*. (n.d.). cs229.stanford.edu/notes/cs229-notes1.pdf

Risk parity. (2018, April 30). en.wikipedia.org/wiki/Risk_parity

Li, F., Chaves, D. B., Hsu, J. C., & Shakernia, O. (2012, July). Efficient Algorithms for Computing Risk Parity Portfolio ... Retrieved June 4, 2018. pdfs.semanticscholar.org/80ea/6bdaba7e654499c8e11ad778dae7970fd29e.pdf