Comparison of Regression Regularizations: A Case Study on Student Grades Data

Jay Wang [*] Yuying Yan [†] Jiaming Zhao[‡]
Jiahui Jiang[§]

May 10, 2019

Abstract

Regression regularization is a powerful technique to avoid overfitting in regression training especially when there are many weights. Trough solving a real-life regression problem, three most popular regularization methods, Ridge, Lasso, and Elastic Net, are fully explored in our study. To fairly evaluate the model efficiency and accuracy, a 5-fold cross-validation scheme is carefully designed; the same cross-validation partitions are used across all training, hyper-parameter tuning, and testing for all of our intersted models. Instead of relying on sophisticated packages and software, we successfully derived a closed-form solution of Ridge regression, and implemented our own coordinate descent algorithms for Lasso regression and Elastic Net. In addition to gaining more insights of how each regularization method works, we have a better appreciation of optimization efficiency. Finally, our results demonstrate that Elastic Net regression outperforms its counterparts while taking more time to find the optimal hyper-parameters.

[*]Department of Statistics Email: zwang688@wisc.edu
[†]Department of Statistics Email: yyan52@wisc.edu
[‡]Department of Statistics Email: jzhao246@wisc.edu
[§]Department of Statistics Email: jjiang86@wisc.edu

1 Introduction

Linear regression is a popular statistical learning method to solve real-life problems, such as house price predictions. This method finds the best linear mapping to approximate the relationships between predictive variable ($X \in \mathbb{R}^{m \times n}$) and the response variable ($Y \in \mathbb{R}^m$).

With the development of computing, people now have a much better capacity to collect, store, and compute data. This leads to a larger amount of features in modern dataset. For example, the predictive variable X has a larger n value. In this case, the traditional simple linear regression, however, tends to find a less generalizable mapping. This caveat is called "overfitting". It happens because this model relies on certain features that could have different distributions in the new data. Therefore, people have come up with methods to avoid "overfitting" by introducing regularizations to penalize models which are over-dependent on certain features. As there is no standard way to choose the best regularization practice, we are interested in comparing all popular approaches and getting a sense of their pros and cons.

To do this, our group have found a real-life dataset featuring Portuguese middle school student attributes and their academic performance. There are many features encoded in this dataset, which makes it appropriate to try out regularization techniques. In addition, the response variable, students' final grades, is a continuous variable. Therefore, we can focus on setting up a regression framework, instead of worrying about logistic regressions directly. Fortunately, the theory of regularizations is transferable from regression to logistic regressions as well.

In our study, we deployed a consistent cross-validation scheme to select hyper-parameters and evaluate different regularizations, so the results can be compared and generalized with minimal bias. Moreover, instead of using sophisticated packages, we implemented our own solvers to solve all these regularized regression problems. This provides us more insights of how and why these penalized regressions work. Our code is included in the Appendix.

2 Dataset

For this project, we choose to use the Student Performance Data Set from https://archive.ics.uci.edu/ml/datasets/student+performance. This dataset contains 395 observations and 30 variables, which include student

grades, demographic, social and school related features. Having multiple variables is essential for our study, because the effect of regularization is only prominent when there are many features in the training data.

The detailed description of this dataset is included in Table 1. There are 4 continuous variables ("age", "absences", "G1", "G2"), and 89 one-hot encoded binary indicators representing the original discrete variables. The "G3" column is used as the response variable Y. Therefore, this dataset gives us a 395×89 sparse matrix to predict a continuous vector with length 395. As the focus of this project is not data analysis, we did not spend extra time to clean or preprocess this dataset.

3 Cross-validation

Since all regularized regression models have hyper-parameter(s), we decided to use a 5-fold cross-validation scheme to train, tune, and test models. All of our models are using the same training, validation, and testing dataset, so the final model comparison is fair.

As described in Fig 1, we first extracted $\frac{1}{5}$ of the data as the hold-out final testing set. Then, on the left $\frac{4}{5}$ of samples, we cross validate on 5 folds to generate 5 training and validation iterations. In each iteration, our model is trained on $\frac{16}{25}$ of original samples and tested on a $\frac{4}{25}$ proportion of all samples. After selecting the best hyper-parameter(s), each model is trained on $\frac{4}{5}$ samples and then tested on the hold-out final testing set.

4 Regression Regularization

4.1 Ridge Regression

Ridge RegressionHoerl and Kennard (1970) is a technique for analyzing multiple regression data that suffer from multicollinearity. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. It is hoped that the net effect will be to give estimates that are more reliable.

Variable	Type	Level/Range	Description
school	Discrete	2	Attended school
sex	Discrete	2	Gender
age	Continuous	15-22	Age
address	Discrete	2	Area of address
famsize	Discrete	2	Family size
Pstatus	Discrete	2	Parent marriage status
Medu	Discrete	5	Mother education
Fedu	Discrete	5	Father education
Mjob	Discrete	5	Mother job
Fjob	Discrete	5	Father job
reason	Discrete	4	Reason to choose this school
guardian	Discrete	3	Students guardian
traveltime	Discrete	5	Travel time between school and home
studytime	Discrete	5	Weekly study time
failures	Discrete	5	Number of failed classes
schoolsup	Discrete	2	Extra education support
famsup	Discrete	2	Family education support
paid	Discrete	2	Extra paid classes
activities	Discrete	2	Extra activities
nursery	Discrete	2	Attended nursery school
higher	Discrete	2	Planned to go to higher education
internet	Discrete	2	Internet access
romantic	Discrete	2	In a romantic relationship
famrel	Discrete	5	Quality of family relationship
freetime	Discrete	5	Free time after school
goout	Discrete	5	Hanging out with friends
Dalc	Discrete	5	Workday alcohol consumption
Walc	Discrete	5	Weekend alcohol consumption
health	Discrete	5	Health status
absences	Continuous	0-93	Number of absences
G1	Continuous	0-20	Exam 1 grade
G2	Continuous	0-20	Exam 2 grade
G3	Continuous	0-20	Final grade

Table 1: Student academic performance dataset.

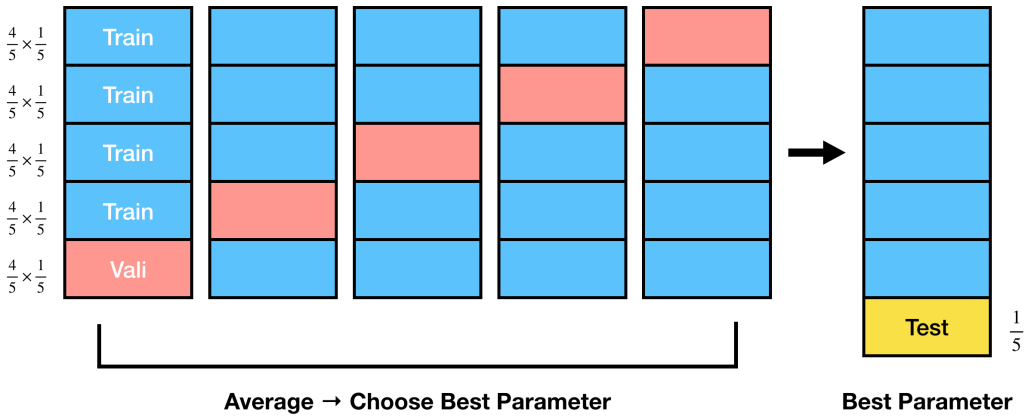

Figure 1: 5-fold Cross-validation to train, tune, and test all of our regularized regression models.

4.1.1 Ridge Regression Cost Function and Partial Derivative

$$\begin{aligned} TotalCost &= RSS(W) + \lambda \|W\|_2^2 \\ &= \sum_{i=1}^{N} (y_i - \sum_{j=0}^{M} w_j x_{ij})^2 + \lambda \sum_{j=0}^{M} w_j^2 \\ &= (y - HW)^T (y - HW) + \lambda W^T W \end{aligned} \tag{1}$$

$$\frac{\partial}{\partial w_j} Cost(W) = -2 \sum_{i=1}^{N} x_{ij} \{ y_i - \sum_{k=0}^{M} w_k x_{ik} \} + 2\lambda w_j \tag{2}$$

$$\nabla Cost(W) = -2H^T (y - HW) + 2\lambda W \tag{3}$$

where W represents coefficients, H is a matrix consisting all observations

4.1.2 L2 Regularization

Ridge Regression performs L2 regularization, which adds a penalty λ when calculating total cost function. Large λ in cost function means high bias, low variance while small λ means low bias, high variance.
Our goal is selecting $\hat{W}$ to minimize the cost function in different choices of

λ and find the optimal λ to perform in our out-of-sample.

4.1.3 Selected $\hat{W}$ brief proof

Separate cost function as two parts:

$$(a) : \sum_{i=1}^{N}(y_i - \sum_{j=0}^{M} w_j x_{ij})^2 = (y - HW)^T(y - HW)$$

$$(b) : \lambda \sum_{j=0}^{M} w_j^2$$

For part(a): *Let* $W^* \in \mathbb{R} \quad \forall t \in [0, 1]$:

$$tf(W)+(1-t)f(W*) = t\sum_{i=1}^{N}(y_i-\sum_{j=0}^{M} w_j x_{ij})^2+(1-t)\sum_{i=1}^{N}(y_i-\sum_{j=0}^{M} w_j^* x_{ij})^2 \quad (1)$$

$$f(tW + (1-t)W^*) = \sum_{i=1}^{N}(y_i - \sum_{j=0}^{M}(tw_j + (1-w_j)x_{ij}))^2 \quad (2)$$

$$(1) - (2) : \sum_{i=1}^{N}\sum_{j=0}^{M}\sum_{k=0}^{M} x_{ij}x_{ik}t(t-1)(w_j - w_j^*)(w_k - w_k^*) \geqslant 0$$

Therefore, part(a) is convex by definition.

For part(b):

The function $f(w) = \lambda\|W\|^2 e$ is 2λ strongly convex by Lemma 13.5 in "Understanding Machine Learning: From Theory to Algorithms."Shalev-Shwartz and Ben-David (2014)

Therefore, the Total Cost function $RSS(W) + \lambda\|W\|_2^2$ is strongly convex by other Lemma in "Understanding Machine Learning: From Theory to Algorithms" that if f is $\lambda-$ strongly convex and g is convex, then $f + g$ is $\lambda-$ strongly convex.

Since the cost function of L2 regularization is a strongly convex function, it has the unique global minimum. We then can use closed-form solution to find selected $\hat{W}$ and calculate Mean Squared Error and R^2 in cross validation

4.1.4 Solve $\hat{W}$

Set the $gradient = 0$, and W is:

$$\begin{aligned} -2H^T(y - HW) + 2\lambda W &= 0 \\ -2H^T(y - HW) + 2I\lambda W &= 0 \\ (H^T H + \lambda I)W &= H^T y \\ W &= (H^T H + \lambda I)^{-1} H^T y \end{aligned} \tag{4}$$

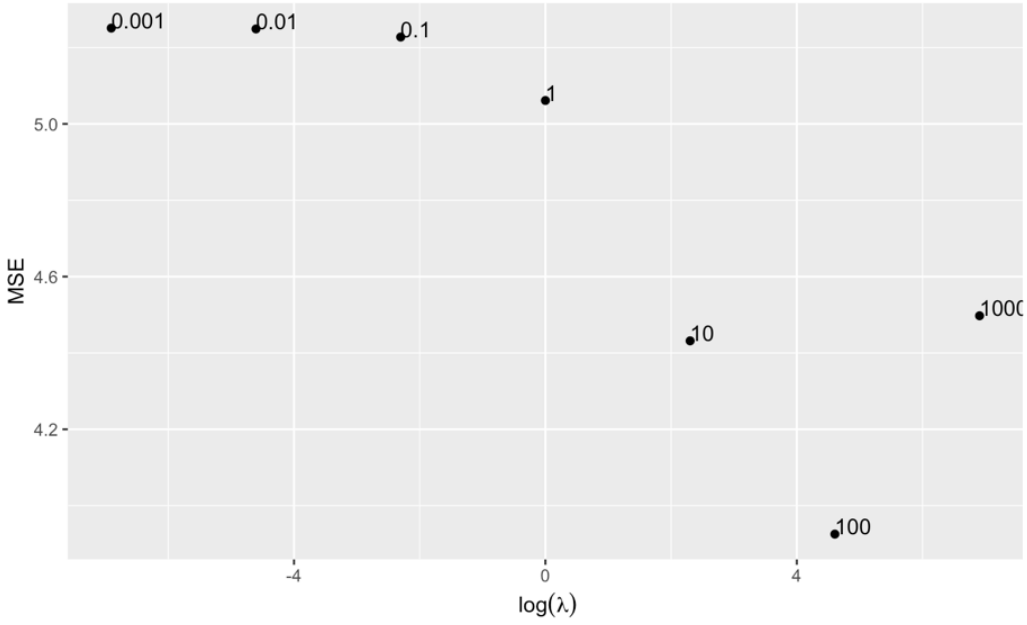

Figure 2: log(Mean Squared Error) calculated under different λ in 5-fold Cross-Validation

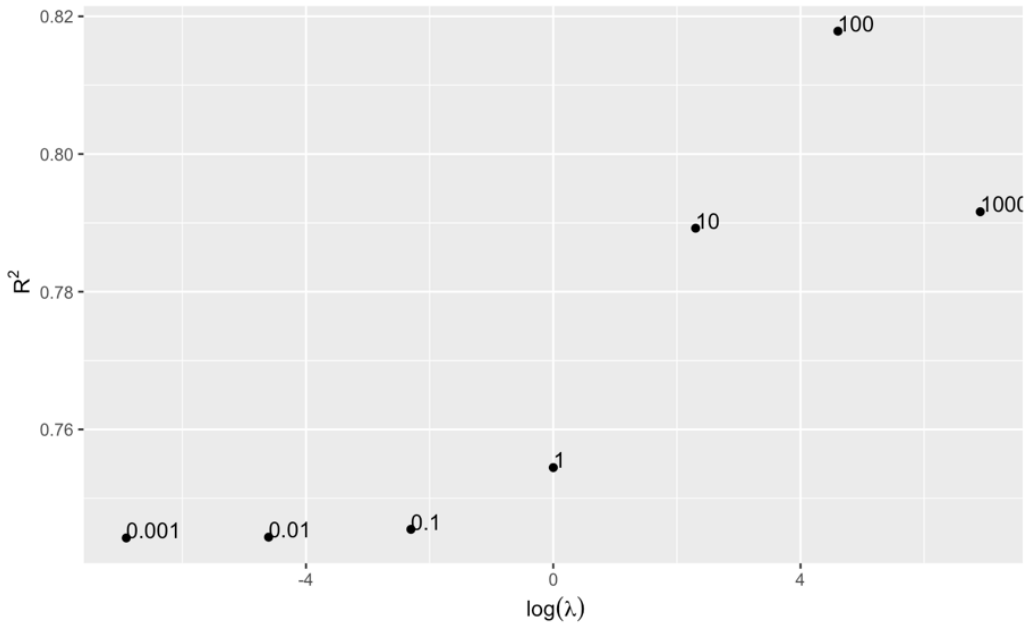

Figure 3: $\log(R^2)$ calculated under different λ in 5-fold Cross-Validation

4.1.5 Result

From the Figure2 and Figure3, obviously $\lambda = 100$ have better results in Mean Squared Error and R^2 when performing cross validation through $\lambda = [0.001, 0.01, 0.1, 1, 10, 100, 1000]$.

Choose $\lambda = 100$ and perform in out-of-sample, we get:

$$Mean\ Squared\ Error = 2.8581 \quad R^2 = 0.8597$$

4.2 Lasso Regression

Lasso (Least Absolute Shrinkage Selector Operator) regressionTibshirani (1996) is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. Lasso encourages simple and sparse models (i.e. models with fewer parameters). Therefore, Lasso is well-suited for models with high levels of muticollinearity or when you want to identify relevant variables in the model. The cost function of Lasso regression is:

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} \left(\frac{1}{2N} \sum_{i=1}^{N} \left(y_i - \beta_0 - x_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \left(|\beta_j|\right) \right)$$

4.2.1 Parameter Shrinkage

The ordinary linear regression minimizes a loss function that contains only RSS, but Lasso adds a shrinking penalty term(a penalty equal to the absolute value of the magnitude of coefficients) to this loss function (As you can see in the equation). So this penalty term has a shrinking effect on the regression coefficient estimate. There is a tuning parameter λ in the penalty term, which controls the strength of the L1 penalty. λ is basically the amount of shrinkage: When $\lambda = 0$, no parameters are eliminated. The loss function is just same as linear regression. As λ increases, more and more coefficients are shrunk to zero and eliminated. Also, when λ increases, bias increases, and when λ decreases, variance increases.

4.2.2 Variable Selection

Lasso is quite similar to ridge, except that Lasso uses L1 norm and it can do both parameter shrinkage and variable selection automatically.When we minimize the optimization problem, some coefficients are shrunk to zero, depending on the value of λ. In this way the variables with coefficient equal to zero are excluded from the model. For this reason Lasso is a powerful method for variable selection while other methods (e.g. Ridge Regression) are not.

4.2.3 L1 Regularization

Lasso regression performs L1 regularization, which adds a shrinking penalty. Lasso regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model. Larger penalties result in coefficient values closer to zero, which is the ideal for producing simpler models. On the other hand, L2 regularization (e.g. Ridge regression) does not result in elimination of coefficients or sparse models. This makes the Lasso far easier to interpret than the Ridge.

4.2.4 Coordinate Descent

Since there is an absolute value in the penalty term, In this case, the gradient is not defined because the absolute function is not differentiable at $\beta = 0$. This can be illustrated as the plot below:

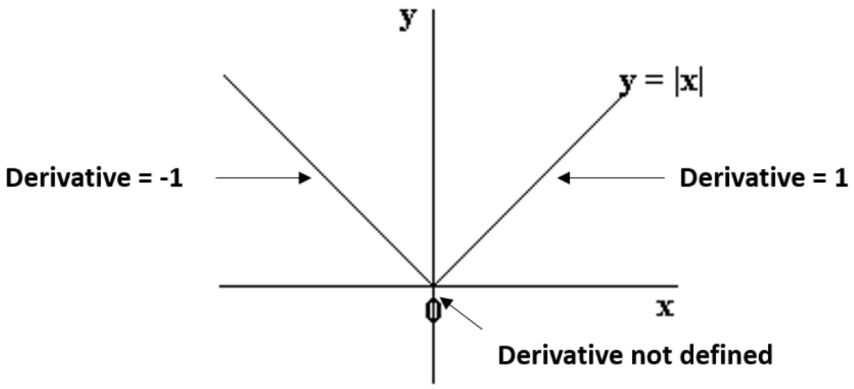

Figure 4: Example of failing to use gradient descent in Lasso

We can see that the parts on the left and right side of 0 are straight lines with defined derivates but the function cant be differentiated at x=0. In this case, we have to use a different technique called coordinate descent which is based on the concept of sub-gradients. The specific algorithm for Lasso coordinate descent is almost identical to the one for Elastic Net except we set α equals to 1.

4.2.5 Result

As a result, the optimal λ we got is 100, the out-of-sample MSE is 2.778, and the out-of-sample R^2 is 0.8636.

The coefficient path plot also indicates that as λ increases, the penalty term will have very strong shrinking effect on the estimate and most variables will be shrunk to very close to 0.

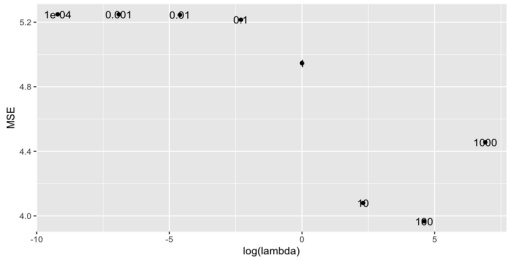

Figure 5: Optimal tuning parameter and the corresponding MSE based on cross-validation.

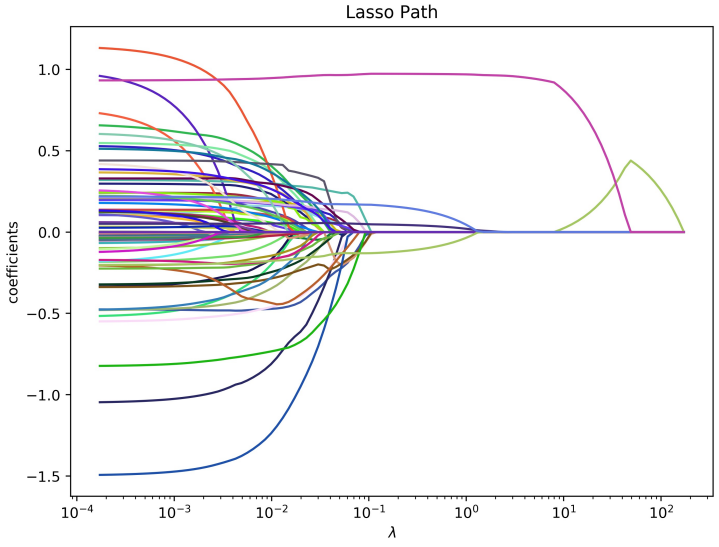

Figure 6: Lasso coefficient path plot

4.3 Elastic Net Regression

4.3.1 Cost Function and Penalty Term

Elastic netZou and Hastie (2005) is simply the combination of Lasso and Ridge regression. The formula is essentially the same except for the penalty term, which is the weighted average of L_1 and L_2 penalty terms. For example, when alpha equals to 0, it becomes Ridge and when alpha equals to 14, it becomes Lasso. The cost function is shown below:

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{n+1}} \left(\frac{1}{2N} \sum_{i=1}^{N} \left(y_i - \beta_0 - x_i^T \beta\right)^2 + \lambda P_\alpha(\beta) \right)$$

The penalty term is:

$$\begin{aligned} P_\alpha(\beta) &= (1-\alpha)\frac{1}{2}\|\beta\|_{\ell_2}^2 + \alpha\|\beta\|_{\ell_1} \\ &= \sum_{j=1}^{p} \left(\frac{1}{2}(1-\alpha)\beta_j^2 + \alpha\left|\beta_j\right| \right) \end{aligned}$$

4.3.2 Coordinate Descent Algorithm Procedure

1. Fix all other coefficients except $beta_1$.
2. Calculate the derivative of cost function with respect to β_1.
3. Set the derivative to 0 and obtain the optimal β_1.
4. Replace the old β_1 with the new one.
5. Repeat the same process to β_2, β_3, etc.

4.3.3 Results

These are the procedures in one iteration. There would be 89 waggles in each iteration. We would run at most 1000 iterations until the difference of results gets smaller than the threshold we set.

Since we have two hyper-parameters (lambda and alpha), there are many combinations of them. These are the plots of MSE and R^2 with respect to different combinations of lambda and alpha.

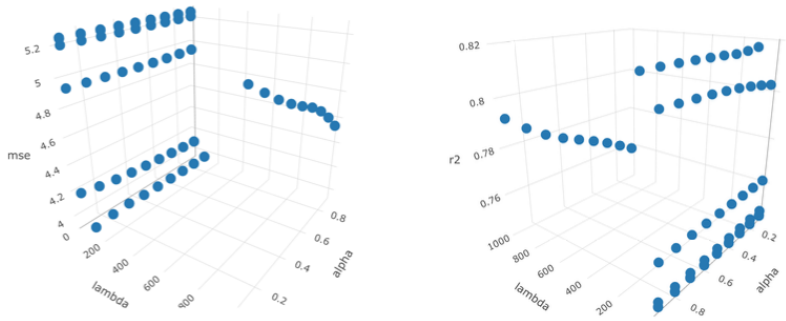

Figure 7: From the plots, we can observe that when lambda equals to 100, the MSE is the smallest and the R^2 gets its highest.

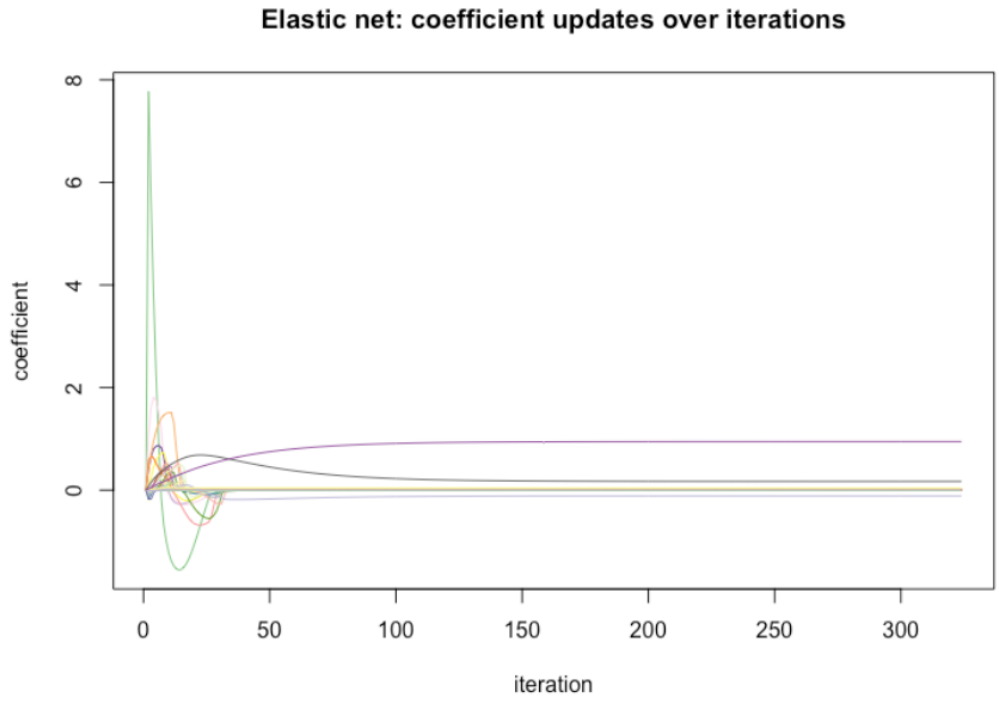

Figure 8: This plot shows the coefficient updates over iterations. Most of the coefficients shrunk and were eliminated eventually. Approximately after 300 iterations, we got our optimal results.

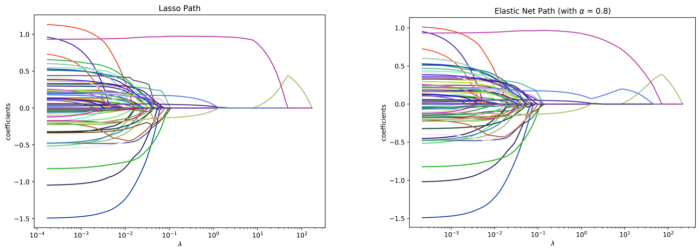

Figure 9: These two plots show the difference of shrinking effects between Lasso and Elastic Net. The shrinking effect of Elastic Net is less aggressive than that of Lasso.

Finally, the optimal alpha and lambda we got according to cross validation results are 0.8 and 100 respectively. The out-of-sample mean squared error is 2.7910 and the out-of-sample R^2 is around 86.3%.

4.4 Comparisons

Model	Hyper-Parameters	MSE	R^2
Original Regression	NA	3.4806	0.8291
Ridge Regression	λ	2.8581	0.8597
Lasso Regression	λ	2.778	0.8636
Elastic Net Regression	α, λ	2.791	0.863

Table 2: Model performance on the same test set.

After training, tuning, and testing all of our four models, we compared the regression prediction performance on the same cross-validation hold-out test set for these models. The results, shown in Table 2, suggest that the performance of penalized regression is significantly better than the original regression. The difference among three regularization methods is marginal, but Lasso and Elastic Net are slightly better than Ridge regression. On the other hand, the time to fit Ridge regression is near instant thanks to the close-form solution. However, the average time of fitting one Lasso regression is 7.81 seconds and 8.84 seconds for Elastic Net regression. The computation time will scale up as the number of parameters increase. Therefore, there is a trade-off between regularization performance and computational cost.

Finally, as we have discussed above, Lasso and Elastic Net both can be used for feature selection, which can be useful for data analysis in some problems.

5 Conclusions

Through this project, we have a better understanding of how and why regularization works for regression problems. In particular, we gained more insights of the nuance between the three most popular methods: Ridge, Lasso, and Elastic Net. On the other hand, we also found that it is critical to use problem-specific optimization techniques to solve numerical problems.

In our study, Jay set up the cross-validation scheme and the universal X and Y to train different models. Yuying derived the close-form solution of Ridge regression, and did model training, tuning, evaluation of Ridge regression. Jiahui implemented the coordinate descent algorithm for Lasso regression, and ran the analysis. Jiaming and Jay implemented the coordinate descent algorithm for Elastic Net and ran the analysis on it.

References

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320.

Appendix: Ridge Regression code

```
##data cleaning

install.packages("mltools")
library(mltools)
library(data.table)

dt1 = read.csv("cv_1.csv")
dt2 = read.csv("cv_2.csv")
dt3 = read.csv("cv_3.csv")
dt4 = read.csv("cv_4.csv")
dt5 = read.csv("cv_5.csv")
test = read.csv("test_set.csv")

dt = rbind(dt1,dt2,dt3,dt4,dt5,test)
grade = data.frame(dt$age,dt$absences,dt$G1,dt$G2,dt$G3)
dt = dt[,c(-3,-9,-10,-30,-31,-32,-33)]


test = dt
onehot = one_hot(as.data.table(test))
onehot_remain = onehot[,c(11,12,20,21,22,39,40,41,42,43,44)]
onehot = onehot[,-c(11,12,20,21,22,39,40,41,42,43,44)]
names = names(onehot_remain)
onehot_remain = as.matrix(onehot_remain)

remain_dummy = data.frame(new = rep(0,nrow(onehot_remain)))

for(i in 1:11){
 r = range(onehot_remain[,i])[1]:range(onehot_remain[,i])[2]
 for(j in 1:length(r)){
  new = rep(0,nrow(onehot_remain))
  new[which(onehot_remain[,i] == r[j])] = 1
  new_dt = data.frame(new)
  names(new_dt) = paste(names[i],r[j])
  remain_dummy = cbind(remain_dummy,new_dt)
 }
}

clean_data = cbind(onehot,remain_dummy[,-1],grade)
write.csv(clean_data[1:63],"cv.1.csv")
write.csv(clean_data[64:126],"cv.2.csv")
write.csv(clean_data[127:189],"cv.3.csv")
write.csv(clean_data[190:252],"cv.4.csv")
```

```
write.csv(clean_data[253:316],"cv.5.csv")
write.csv(clean_data[317:395],"test.csv")

##cross validation

cv.1 = read.csv("cv.1.csv")[,-1]
cv.2 = read.csv("cv.2.csv")[,-1]
cv.3 = read.csv("cv.3.csv")[,-1]
cv.4 = read.csv("cv.4.csv")[,-1]
cv.5 = read.csv("cv.5.csv")[,-1]

lambda =c(1000,100,10, 1, 0.1, 0.01, 0.001)

#calculate coefficient
cal_coef = function(X,lambda,y){
  ##closed form solution
  ##set the graident = 0
  X = X
  n_lambda = lambda
  part.a = (t(X)%*%X + n_lambda*diag(90))
  inv.a = solve(part.a)
  coefficient = inv.a %*% t(X) %*% y
  return(coefficient)
}

## use coefficients to estimate test sets
## estimate error
cal_mse = function(test.set,coef){
  dt = test.set
  y = dt[,ncol(dt)]
  X = as.matrix(cbind(rep(1,nrow(dt)),dt[,-ncol(dt)]))
  est_y = X %*% coef
  mse = sum((y-est_y)^2)/nrow(dt)
  return(mse)
}

cal_rsquare = function(test.set,coef){
  dt = test.set
  y = dt[,ncol(dt)]
  y_bar = mean(y)
  X = as.matrix(cbind(rep(1,nrow(dt)),dt[,-ncol(dt)]))
  est_y = X %*% coef
  tot = sum((y-y_bar)^2)
  res = sum((y-est_y)^2)
  rsquare = 1-res/tot
```

```
  return(rsquare)
}

cv_func = function(tr1,tr2,tr3,tr4,test,lambda){
  train.dt = rbind(tr1,tr2,tr3,tr4)
  test.dt = test

  y = train.dt[,90] #g3 grade
  X = as.matrix(cbind(rep(1, nrow(train.dt)),train.dt[,-90]))

  coef = cal_coef(X,lambda,y)
  mse = cal_mse(test.dt,coef)
  rsquare = cal_rsquare(test.dt,coef)

  return(list(coef = coef, MSE = mse, Rsquare = rsquare))
}

record_mse = matrix(0,6,length(lambda))#record
record_rs =  matrix(0,6,length(lambda))
for(i in 1:length(lambda)){
  ##each calculated mse
  m1 = cv_func(cv.1,cv.2,cv.3,cv.4,cv.5,lambda[i])
  m2 = cv_func(cv.1,cv.2,cv.3,cv.5,cv.4,lambda[i])
  m3 = cv_func(cv.1,cv.2,cv.5,cv.4,cv.3,lambda[i])
  m4 = cv_func(cv.1,cv.5,cv.3,cv.4,cv.2,lambda[i])
  m5 = cv_func(cv.5,cv.2,cv.3,cv.4,cv.1,lambda[i])
  ##do above procedure 5 times and get average error
  record_mse[,i] = c(m1$MSE,m2$MSE,m3$MSE,m4$MSE,m5$MSE,
  mean(c(m1$MSE,m2$MSE,m3$MSE,m4$MSE,m5$MSE)))
  record_rs[,i] = c(m1$Rsquare,m2$Rsquare,m3$Rsquare,m4$Rsquare,m5$Rsquare,
  mean(c(m1$Rsquare,m2$Rsquare,m3$Rsquare,m4$Rsquare,m5$Rsquare)))
}

colnames(record_mse) = c(lambda)
rownames(record_mse) = c(1:5,"average")
colnames(record_rs) = c(lambda)
rownames(record_rs) = c(1:5,"average")

record_mse
record_rs


##plot

library(ggplot2)
```

```
lm = rep(log(lambda, exp(1)),1)
cv.test = c("average","average","average","average","average","average","average")
notation = lambda

mse = c(record_mse[6,])
plot_dt = data.frame(mse = mse, lambda =lm, notation = lambda)

ggplot(data = plot_dt, aes(x = lambda, y = mse))+geom_point()+
labs(y ="MSE",x = bquote(~log(lambda)))+geom_text(aes(label=notation),hjust=0, vjust=0)

rsq = c(record_rs[6,])
plot_dt_rs = data.frame(r_square = rsq, cv = cv.test, lambda =lm)
ggplot(data = plot_dt_rs, aes(x = lambda, y =r_square))+geom_point()+
labs(y =bquote(~R^2),x = bquote(~log(lambda)))+geom_text(aes(label=notation),hjust=0, vjust=0)
```

Appendix: Lasso Regression code

```
def lasso_train(x, y, alpha=1, lamb=0.1, max_iter=1000, epsilon=1e-6):

    # Add a column for intercept coef
    train_x = np.column_stack((np.ones(len(x)), x))

    # Just initialize all coefs to 0
    weights = np.zeros(train_x.shape[1])

    # Compute c_1, c_2 (it is same for every j)
    #c_1 = 2 * lamb * (1 - alpha)
    c_2 = lamb * alpha

    # No converge stopping
    for iter in range(max_iter):
        old_weights = weights.copy()

        # Iterate through all coefficients
        for j in range(len(weights)):

            # Exclude the contribution of current coefficient
            cur_weights = weights.copy()
            cur_weights[j] = 0

            # Compute p_j (1): residuals (without contribution of w_j)
            predict = np.dot(train_x, cur_weights)
            residual = y - predict
```

```
            # Compute p_j (2): weight residuals by x_j and take sum (dot product)
            p_j = np.dot(train_x[:, j], residual)

            # Compute z_j
            z_j = np.sum(train_x[:, j] ** 2)

            # Compare p_j with alpha * lambda and update weights
            if p_j < -c_2:
                weights[j] = (p_j + c_2) / (z_j)
            elif p_j > c_2:
                weights[j] = (p_j - c_2) / (z_j)
            else:
                weights[j] = 0

        # Need to specially handle the intercept (best intercept)
        best_intercepts = y - np.dot(train_x[:, 1:], weights[1:])
        weights[0] = np.sum(best_intercepts) / (train_x.shape[0])

        # If the update is too small, we stop the iteration
        max_update = np.max(np.abs(weights - old_weights))
        if max_update < epsilon:
            # print("Early stop: at iteration {}".format(iter))
            return weights

    return weights


def lasso_predict(x, coefs):
    x_padding = np.column_stack((np.ones(len(x)), x))
    y_predict = np.dot(x_padding, coefs)
    return y_predict

start_time = time.time()
error_list = {}
for l in lamb_list:
    errors = 0

    # for each fold of cross validation
    for i in range(len(train_dict)):

        #use train set to tune weight
        weight = lasso_train(train_dict[i]['train_x'],train_dict[i]['train_y'],lamb = l)

        #use vali set to test accuracy
        # MSE = sum((y-y.pred)**2)/n
```

```
        error = np.average((vali_dict[i]['vali_y']-lasso_predict(vali_dict[i]['vali_x'],weight))
        errors = errors+error
    errors = errors/len(train_dict)
    error_list[l] = errors
print("--- %s seconds ---" % (time.time() - start_time))

test_weight = lasso_train(test_train['train_x'],test_train['train_y'],lamb = 100)

test_y = test_test['test_y']
test_x = test_test['test_x']
E_test_y = np.average(test_y)
E_test_y_array = np.empty(test_y.shape[0])
E_test_y_array.fill(E_test_y)

SSt = np.sum((test_y - E_test_y_array)**2)
SSr = np.sum((test_y-lasso_predict(test_x,test_weight))**2)
#MSE
MSE = SSr/test_y.shape[0]

#R^2
R_square = 1-SSr/SSt
```

Appendix: ElasticNet Regression code

```
def en_train(x, y, alpha=0.5, lamb=0.1, max_iter=1000, epsilon=1e-6):

    coef_track = []

    # Add a column for intercept coef
    train_x = np.column_stack((np.ones(len(x)), x))

    # Just initialize all coefs to 0
    weights = np.zeros(train_x.shape[1])

    # Compute c_1, c_2 (it is same for every j)
    c_1 = 2 * lamb * (1 - alpha)
    c_2 = lamb * alpha

    # No converge stopping
    for iter in range(max_iter):
        old_weights = weights.copy()

        # Iterate through all coefficients
        for j in range(len(weights)):
```

```
            # Exclude the contribution of current coefficient
            cur_weights = weights.copy()
            cur_weights[j] = 0

            # Compute p_j (1): residuals (without contribution of w_j)
            predict = np.dot(train_x, cur_weights)
            residual = y - predict

            # Compute p_j (2): weight residuals by x_j and take sum (dot product)
            p_j = np.dot(train_x[:, j], residual)

            # Compute z_j
            z_j = np.sum(train_x[:, j] ** 2)

            # Compare p_j with alpha * lambda and update weights
            if p_j < -c_2:
                weights[j] = (p_j + c_2) / (z_j + c_1)
            elif p_j > c_2:
                weights[j] = (p_j - c_2) / (z_j + c_1)
            else:
                weights[j] = 0

        # Need to specially handle the intercept (best intercept)
        best_intercepts = y - np.dot(train_x[:, 1:], weights[1:])
        weights[0] = np.sum(best_intercepts) / (train_x.shape[0])

        coef_track.append(weights.tolist())

        # If the update is too small, we stop the iteration
        max_update = np.max(np.abs(weights - old_weights))
        if max_update < epsilon:
            # print("Early stop: at iteration {}".format(iter))
            return (weights, coef_track)

    return (weights, coef_track)


def en_predict(x, coef):
    x_padding = np.column_stack((np.ones(len(x)), x))
    y_predict = np.dot(x_padding, coefs)
    return y_predict
```