PES University, Bengaluru
UE18CS312 - Data Analytics

Session: Aug – Dec 2020
Weeks 3-4 – Code Snippets for Worksheet 2(b) (for Unit 2)

Compiled by: Ms. Richa and Ms. Mainaki Saraf
VII CSE, PES University RR Campus

3.0 Getting Started

The "Hitters" dataset which is a part of R's ISLR package will be used for this worksheet. The dataset is also available at
https://gist.github.com/keeganhines/59974f1ebef97bbaa44fb19143f90bad .

1. Read this dataset
   Install the ISLR library which has the dataset or download from the above link and read the file as a csv. `library(ISLR)`

2. Remove/Fill in the missing values using a suitable technique
   For convenience, one can omit the missing values. You can try out different filing methods too if you want. The solution given is obtained by removing all missing values.
   `Hitters = na.omit(Hitters)`

3. The main motive is to build a regression model using ridge and lasso regression techniques to predict the salary of a hitter given all the other variables.

4. Separate the dependent and independent variables.
   Y = Salary X = all other variables in the dataset

```
x = model.matrix(Salary~., Hitters)[,-1]
y = Hitters %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()
```

Model matrix is useful for X, as it transforms qualitative variables into dummy variables which is an important property needed during the application of ridge/lasso regression models.

3.1 Ridge Regression

1. Explore the **glmnet** library in R for this exercise.
   The glmnet library has the glmnet() to implement ridge/lasso regression.

```
library(glmnet)
ridge_mod = glmnet(x, y)
```

2. Alter the arguments on the glmnet() to fit a ridge regression model to the data. Which argument has to be changed and what should it be set to implement this model?

The value of the alpha argument in the glmnet() decides which model will be fit to the data. If alpha = 0, a ridge regression model is fit. If alpha =1, then a lasso regression model is fit.

```
ridge_mod = glmnet(x, y, alpha = 0)  //ridge
ridge_mod = glmnet(x, y, alpha = 1)  //lasso
```

3. Are the variable values standardized? What can be done to avoid the standardization? Hint: **Check glmnet arguments**

By default, glmnet() standardizes the values. This can be avoided by setting the argument, standardize = FALSE

4. How does the model select the values for ƛ? Is it possible to restrict the possible values to a particular range?

By default the glmnet() function performs ridge regression for an automatically selected range of λ values. It is possible to implement the function over a grid of values. Eg: ranging from 10^10 to 10^-2, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit.

```
grid = 10^seq(10, -2, length = 100)
ridge_mod = glmnet(x, y, alpha = 0, lambda = grid)
```

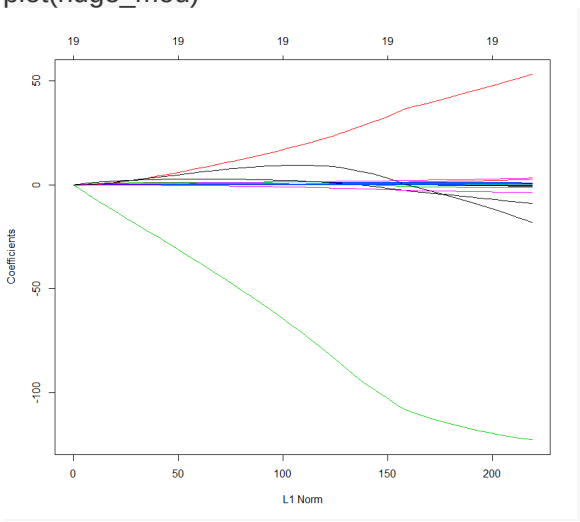
5. Try retrieving the coefficient estimates. In what form are they? What is the shape of the same? When are the estimates expected to be of a larger value? When are they of smaller values?

Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by coef(). In this case, it is a 20×100 matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ).

```
dim(coef(ridge_mod))
```

```
> dim(coef(ridge_mod))
[1]  20 100
```

```
plot(ridge_mod)
```



L2 Norm

We expect the coefficient estimates to be much smaller, in terms of L2 norm, when a large value of λ is used, as compared to when a small value of λ is used.

6. Split the data into test and train sets and fit the model with the training set. Predict the values for the test set and record the MSE value.

Splitting into test and train sets:

```
train = Hitters %>%
  sample_frac(0.5)

test = Hitters %>%
  setdiff(train)

x_train = model.matrix(Salary~., train)[,-1]
x_test = model.matrix(Salary~., test)[,-1]

y_train = train %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()

y_test = test %>%
  select(Salary) %>%
  unlist() %>%
  as.numeric()
```

Model fitting and prediction using lambda value i.e s = 4 (randomly chosen value)

```
ridge_mod = glmnet(x_train, y_train, alpha=0, lambda = grid, thresh = 1e-12)
ridge_pred = predict(ridge_mod, s = 4, newx = x_test)
mean((ridge_pred - y_test)^2)
```

7. Try changing the lambda value used for prediction to see how the MSE value varies. What will the value of lambda be if the model was to replicate a least-square fit?

The best value of lambda will be the one which gives the least MSE. A validation set can be made to get this best lambda value and then this value can be used to predict for the test set.

Or use,

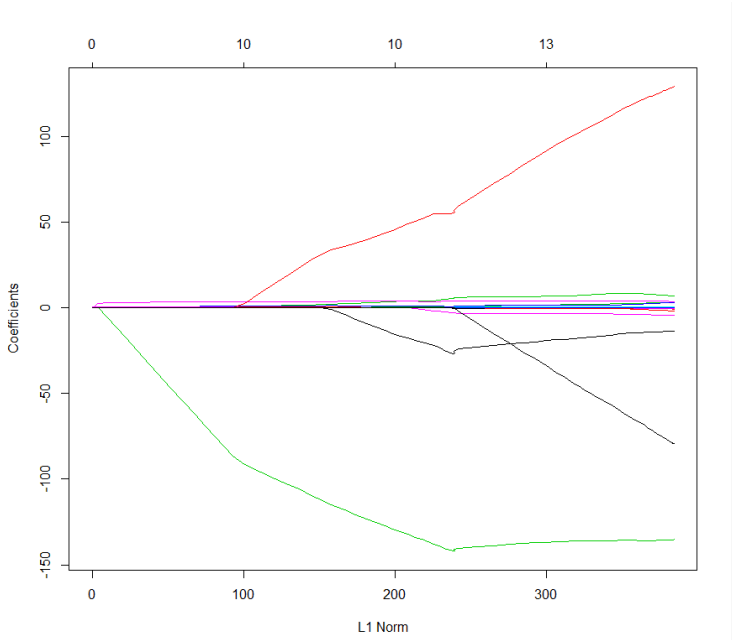
```
cv.out = cv.glmnet(x_train, y_train, alpha = 0)
bestlam = cv.out$lambda.min
```

In order for glmnet() to yield the exact least squares coefficients when $\lambda=0$, we use the argument exact=T when calling the predict() function. Otherwise, the predict() function will interpolate over the grid of λ values used in fitting the glmnet() model, yielding approximate results. Even when we use exact = T, there remains a slight discrepancy in the third decimal place between the output of glmnet() when $\lambda=0$ and the output of lm(); this is due to numerical approximation on the part of glmnet().

3.2 Lasso Regression

1. From the above exercise of ridge regression fit a lasso regression model to the train set. Hint: **Make changes in the parameter of glmnet()**

```
lasso_mod = glmnet(x_train, y_train, alpha = 1)
plot(lasso_mod)
```



Notice that in the coefficient plot that depending on the choice of tuning parameter, some of the coefficients are exactly equal to zero.

2. Predict the values for the test set. Calculate the MSE and compare to that of the ridge regression model with the same lambda value.

Model Fitting and Prediction:

```
cv.out = cv.glmnet(x_train, y_train, alpha = 1)  # Fit lasso model on training data
plot(cv.out)                                     # Draw plot of training MSE as a function of lambda
bestlam = cv.out$lambda.min    # Select lambda that minimizes training MSE
lasso_pred = predict(lasso_mod, s = bestlam, newx = x_test) # Use best lambda to predict test data
mean((lasso_pred - y_test)^2) # Calculate test MSE
```

3. Try printing the predictors which were kept by the model and the ones which were discarded.

```
out = glmnet(x, y, alpha = 1, lambda = grid)   # Fit lasso model on full dataset
lasso_coef = predict(out, type = "coefficients", s = bestlam)[1:20,]  # Display coefficients using lambda chosen by CV
lasso_coef
```

The lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 2 of the 19 coefficient estimates are exactly zero.

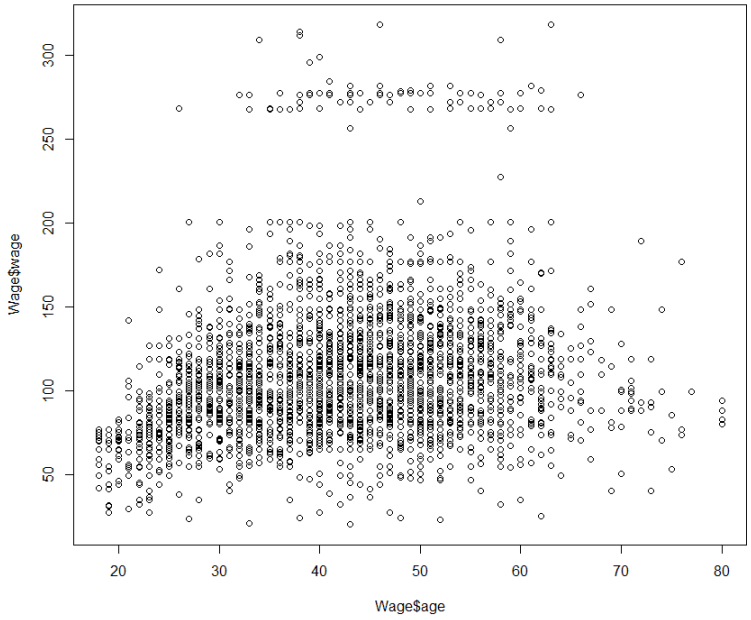
4.1 Polynomial Regression

For this part of the worksheet, the “Wages” dataset in R will be used. It is also available at https://www.picostat.com/dataset/r-dataset-package-islr-wage
Read the dataset to the file. The model which will be built will predict the wage of a person given certain input variables.

1. Plot the “Age” and “Wage” columns and observe their relationship. What kind of a relationship is it?

   Following is the plot for age vs wage.

   plot(Wage$wage ~ Wage$age)



   As it can be seen, the relationship is non linear in nature and a straight line can’t be used to fit the data. A parabola or a polynomial of a higher degree may fit the data better in this case as compared to a straight line.

2. Taking only the “Age” column as input. Try fitting a linear regression model. Now, change the degree of this column to 2 and fit another linear regression model. Hint: Explore **poly()**. What does poly() return? Determine the new equation for the model.

   Linear Regression Model:

   ```
   fit = lm(wage ~ poly(age, 1), data = Wage)
   ```

   Changing the degree to 2:

   ```
   fit = lm(wage ~ poly(age, 2), data = Wage)
   ```

   The poly() command allows us to avoid having to write out a long formula with powers of age. The function returns a matrix whose columns are a basis of orthogonal polynomials, which essentially means that if the degree is 4, then each column is a linear combination of the variables age, age^2, age^3 and age^4.

   For Degree = 1:

```
> fit = lm(wage ~ poly(age, 1), data = Wage)
> coef(summary(fit))
                Estimate Std. Error   t value     Pr(>|t|)
(Intercept)    111.7036  0.7472592 149.48442 0.000000e+00
poly(age, 1) 447.0679 40.9290707  10.92299 2.900778e-27
```

   Equation: Wage = 111.7036 + 447.0679(Age)

   For Degree = 2:

```
> fit = lm(wage ~ poly(age, 2), data = Wage)
> coef(summary(fit))
                Estimate Std. Error   t value     Pr(>|t|)
(Intercept)    111.7036   0.730162 152.98470 0.000000e+00
poly(age, 2)1  447.0679  39.992617  11.17876 1.878131e-28
poly(age, 2)2 -478.3158  39.992617 -11.96010 3.077420e-32
```

Equation: Wage = 111.7036 + 447.0679(Age) - 478.3158(Age^2)

3. Keep increasing the degree upto 5 and play around with different degrees and see how the model behaves. Observe the changes in the coefficients and the intercept. Add more variables as you wish and see how the model equation changes. Experiment with different degrees and combinations!

4. Explore other methods to compare how the various models with different degrees performed. Hint: **compare p-values using anova()** for all the fitted models. Which model provides a reasonable fit based on this test (with only age as input)? Give reasons.

Method 1: Compare p-values using ANOVA for the various fits.

```
fit_1 = lm(wage~age, data = Wage)
fit_2 = lm(wage~poly(age,2), data = Wage)
fit_3 = lm(wage~poly(age,3), data = Wage)
fit_4 = lm(wage~poly(age,4), data = Wage)
fit_5 = lm(wage~poly(age,5), data = Wage)
print(anova(fit_1,fit_2,fit_3,fit_4,fit_5))
```

```
  Res.Df     RSS Df Sum of Sq        F    Pr(>F)
1   2998 5022216
2   2997 4793430  1    228786 143.5931 < 2.2e-16 ***
3   2996 4777674  1     15756   9.8888  0.001679 **
4   2995 4771604  1      6070   3.8098  0.051046 .
5   2994 4770322  1      1283   0.8050  0.369682
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The p-value comparing the linear Model 1 to the quadratic Model 2 is essentially zero, indicating that a linear fit is not sufficient. Similarly the p-value comparing the quadratic Model 2 to the cubic Model 3 is very low , so the quadratic fit is also insufficient. The p-value comparing the cubic and degree-4 polynomials, Model 3 and Model 4, is approximately 0.05 while the degree-5 polynomial Model 5 seems unnecessary because its p-value is high (0.36). Hence, either a cubic or a quartic polynomial appears to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

This is one way, any other valid approach to this question is welcome. Like error comparisons, R2 scores, splitting into test and train sets and comparing, etc.

4.2 Logistic Regression

For this part of the worksheet the “Smarket” dataset from R’s ISLR library will be used. Its also available at https://github.com/selva86/datasets/blob/master/Smarket.csv
Read the dataset to the file. A logistic regression model is to be fit to predict the Direction using Lag1 to Lag5 and the Volume variable.

1. Split the data into test and train sets. How will you perform the split? Will a random split give the required results? If not random, then which approach will you use? Hint: Look into the **Year** column

```
train = Smarket %>%
  filter(Year < 2005)

test = Smarket %>%
  filter(Year >= 2005)
```

We fit a logistic regression model using only the subset of the observations that correspond to dates before 2005, using the subset argument. We then obtain predicted probabilities of the stock market going up for each of the days in our test set—that is, for the days in 2005.

2. Try fitting a logistic regression model for the target variable by giving the above-mentioned variables as input for the training set. Hint: Explore **glm()**

Which argument is altered to fit the logistic regression model and what is the value given?

```
glm_fit = glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume, data = train, family = binomial)
```

The argument family is set to binomial to fit a logistic regression model.

3. Get a summary of the model built. Get the coefficients and residuals too and observe how different input variables play a role in determining the target value.

```
summary(glm_fit)
coefficients(glm_fit)
residuals(glm_fit)
```

Summary Output:

```
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
    Volume, family = binomial, data = train)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.302  -1.190   1.079   1.160   1.350

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.191213   0.333690   0.573    0.567
Lag1        -0.054178   0.051785  -1.046    0.295
Lag2        -0.045805   0.051797  -0.884    0.377
Lag3         0.007200   0.051644   0.139    0.889
Lag4         0.006441   0.051706   0.125    0.901
Lag5        -0.004223   0.051138  -0.083    0.934
Volume      -0.116257   0.239618  -0.485    0.628

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1383.3  on 997  degrees of freedom
Residual deviance: 1381.1  on 991  degrees of freedom
AIC: 1395.1
```

All the p values are high, this may be because the sample size is too small.

4. Try predicting the values for the test set. In which form are the predictions? What can be done to get the class labels instead as the predicted output? Hint: **Set a threshold** on the output value to classify it as one class.

```
glm_probs = data.frame(probs = predict(glm_fit, newdata = test, type="response"))
```

The predictions are in the form of probabilities. To get the class labels for the output, a threshold of 0.5 can be set to segregate into the two classes as follows and added as a column to the result dataframe. If probability > 0.5 , then its “Up”, otherwise its “Down”.

```
glm_pred = glm_probs %>%
  mutate(pred = ifelse(probs>.5, "Up", "Down"))   //transform result based on probability

glm_pred = cbind(test, glm_pred)   //add column to main dataframe
```

5. Check how the model performed for the test set. Which metric/metrics will you use to evaluate the model’s performance?

Confusion matrices can be plotted to get a good understanding as to how the classifier performed. Furthermore based on these values, accuracy score, precision, recall, etc can be computed to see how the classifier did. ROC can be plotted to visualize the fit of the classifier.

For confusion matrix:

```
glm_pred %>%
  count(pred, Direction) %>%
  spread(Direction, n, fill = 0)
```

```
# A tibble: 2 x 3
  pred   Down    Up
  <chr> <dbl> <dbl>
1 Down     77    97
2 Up       34    44
```

For Accuracy Score:

```
glm_pred %>%
  summarize(score = mean(pred == Direction))
```

Score gives the ones predicted correctly i.e the accuracy.

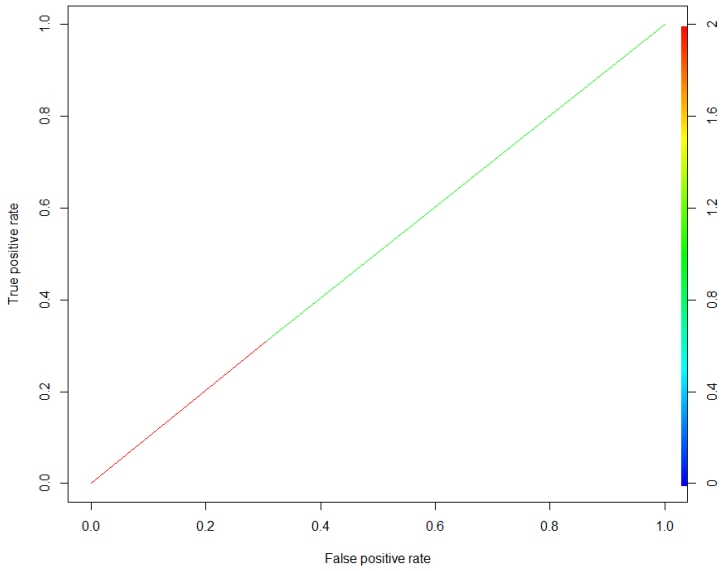
```
> glm_pred %>%
+   summarize(score = mean(pred == Direction))
      score
1 0.4801587
```

As the accuracy is 48%, i.e the model is a bad classifier. Using the confusion matrix above, other metrics can also be calculated to support the statement.

Plotting the ROC Curve:
Install the ROCR package.
Convert the “UP” and “DOWN” labels in the results dataframe (glm_pred) to 1 and 0 respectively and run the following lines of code. ROCR only supports continuous labels.

```
pred <- prediction(glm_pred$pred,glm_pred$Direction)
perf <- performance(pred,"tpr","fpr")
plot(perf,colorize=TRUE)
```

ROC CURVE



The plot is a 45 degree line showing the classifier is a bad classifier.