Applied Statistical Methods

Linear Model Regularization - Part I

Xuemao Zhang
East Stroudsburg University

March 31, 2023

Outline

- Ridge Regression
- Lasso Regression

Introduction

- The subset selection methods use least squares to fit a linear model that contains a subset of the predictors.
- As an alternative, we can fit a model containing all p predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.
- This is known as regularization or penalization.
- It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.

Introduction

Crazy Coefficients

- When $p > n$, some of the variables are highly correlated.
- Why does correlation matter?
  - Suppose that X_1 and X_2 are highly correlated with each other... assume $X_1 = X_2$ for the sake of argument.
  - And suppose that the least squares model is
  $$\hat{y} = X_1 - 2X_2 + 3X_3$$
  - Then this is also a least squares model (see a simulation study next slide):
  $$\hat{y} = 100000001X_1 - 100000002X_2 + 3X_3$$
- Bottom Line: When there are too many variables, the least squares coefficients can get crazy!
- This craziness is directly responsible for poor test error.
- It amounts to too much model complexity.

Introduction

Crazy Coefficients

```
import numpy as np
from numpy import random
from scipy.stats import pearsonr
import pandas as pd
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols
random.seed(1)
x1=random.normal(0, 15, size=20)
x2=np.zeros(20)
for i in range(20):
    x2[i]=random.normal(x1[i], 0.001, size=1)
pearsonr(x1, x2)
```

```
## PearsonRResult(statistic=0.999999998813472, pvalue=4.426361402320335e-79)
```

```
x3=random.uniform(size=20)
y=np.zeros(20)
for i in range(20):
    y[i]=random.normal(-x1[i]+3*x3[i], 1,size=1)
data=pd.DataFrame({'y':y, 'x1':x1, 'x2':x2, 'x3':x3})
fit=smf.ols('y~x1+x2+x3', data).fit()
#fit=smf.ols('y~x1+x2+x3-1', data).fit() #no intercept
print(fit.params)
```

```
## Intercept     -0.268871
## x1          -155.080603
## x2           154.104405
## x3             3.989046
## dtype: float64
```

Ridge Regression

- Recall that the least squares fitting procedure estimates $\beta_0, \beta_1, \ldots, \beta_p$ using the values that minimize

$$RSS = \|\boldsymbol{y} - \boldsymbol{X}\beta\|^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

- In contrast, the ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

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

where $\lambda > 0$ is a tuning parameter, to be determined separately.

Ridge Regression

- Equivalently, we find $\hat{\beta}^R$ that minimizes

$$\|\boldsymbol{y} - \boldsymbol{X}\beta\|^2$$

subject to the constraint that

$$\sum_{j=1}^{p} \beta_j^2 < s$$

for some s.

Ridge Regression

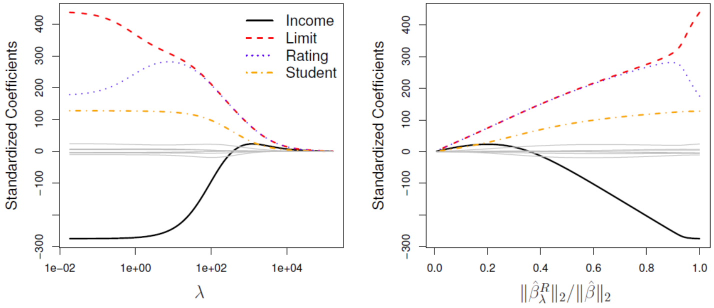
- Ridge regression coefficient estimates minimize

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

- When $\lambda = 0$, then ridge regression is just the same as least squares.
- As λ increases, then $\sum_{j=1}^{p} (\hat{\beta}_{\lambda,j}^R)^2$ decreases - i.e. coefficients become shrunken towards zero.
- As $\lambda \to \infty$, $\hat{\boldsymbol{\beta}}^R = 0$.

Ridge Regression

- Ridge Regression As λ Varies: The standardized ridge regression coefficients are displayed for the Credit data set.



Ridge Regression

Ridge regression: scaling of predictors

- The standard least squares coeffcient estimates are scale equivariant: multiplying X_j by a constant c simply leads to a scaling of the least squares coefficient estimates by a factor of $1/c$. In other words, regardless of how the jth predictor is scaled, $X_j\hat{\beta}_j$ will remain the same.
- In contrast, the ridge regression coefficient estimates can change substantially when multiplying a given predictor by a constant, due to the sum of squared coefficients term in the penalty part of the ridge regression objective function.
- Therefore, it is best to apply ridge regression after standardizing the predictors, using the formula

$$\tilde{x}_{ij} = \frac{x_{ij}}{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}, i = 1, \dots, n, j = 1, \dots, p$$

Ridge Regression In Practice

- Perform ridge regression for a very fine grid of λ values.
- Use cross-validation or the validation set approach to select the optimal value of λ - that is, the best level of model complexity.
- Perform ridge on the full data set, using that value of λ.

Drawbacks of Ridge

- Ridge regression is a simple idea and has a number of attractive properties: for instance, you can continuously control model complexity through the tuning parameter λ.
- But it suffers in terms of model interpretability, since the final model contains all p variables, no matter what.
- We Often want a simpler model involving a subset of the features.
- The lasso involves performing a little tweak to ridge regression so that the resulting model contains mostly zeros.
- In other words, the resulting model is sparse. We say that the lasso performs feature selection.

Lasso Regression

- The lasso involves finding β that minimizes

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

  where $\lambda > 0$ is a tuning parameter.

- Equivalently, we find $\hat{\beta}^L$ that minimizes

$$\|\boldsymbol{y} - \boldsymbol{X}\beta\|^2$$

  subject to the constraint that

$$\sum_{j=1}^{p} |\beta_j| < s$$

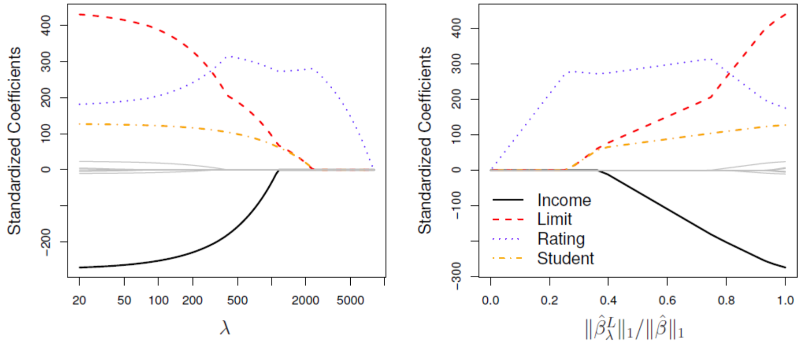
  for some s.

Lasso Regression

- Lasso is a lot like ridge:
  - λ is a nonnegative tuning parameter that controls model complexity.
  - When $\lambda = 0$, we get least squares.
  - When λ is very large, we get $\hat{\beta}^L = 0$.
- But unlike ridge, lasso will give some coefficients exactly equal to zero for intermediate values of λ!
- Hence, much like best subset selection, the lasso performs variable selection.
- We say that the lasso yields sparse models - that is, models that involve only a subset of the variables.

Lasso Regression

- Lasso Regression As λ Varies: The Lasso regression coefficients are displayed for the Credit data set.



Lasso Regression In Practice

- Perform lasso for a very fine grid of λ values.
- Use cross-validation or the validation set approach to select the optimal value of λ - that is, the best level of model complexity.
- Perform the lasso on the full data set, using that value of λ.

Ridge and Lasso: A Geometric Interpretation

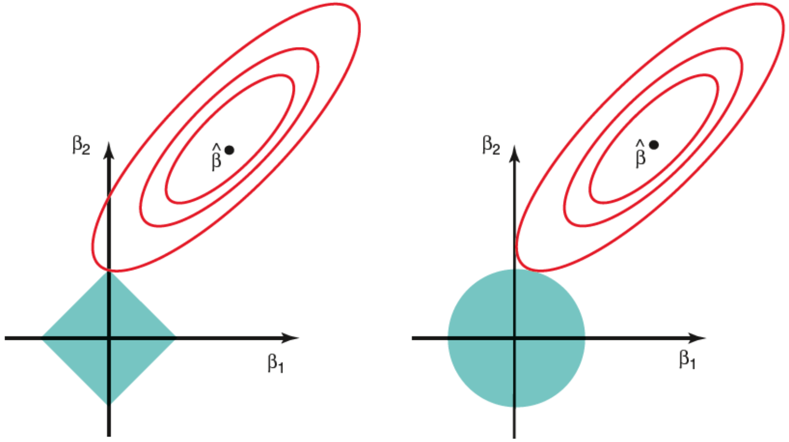


FIGURE *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \le s$ *and* $\beta_1^2 + \beta_2^2 \le s$, *while the red ellipses are the contours of the RSS.*

Python: Ridge Regression

- We predict Salary on the Hitters data

```
import pandas as pd
Hitters=pd.read_csv("../data/Hitters.csv", header=0, na_values='NA')
Hitters = Hitters.dropna()
dummies = pd.get_dummies(Hitters[['League', 'Division', 'NewLeague']])
y = Hitters['Salary']
X_ = Hitters.drop(['Salary', 'League', 'Division', 'NewLeague'],
axis=1).astype('float64')
X = pd.concat([X_, dummies[['League_N', 'Division_W', 'NewLeague_N']]],
axis=1)
```

Python: Ridge Regression

- We will use the sklearn package in order to perform ridge regression and the lasso regression.
  - The `Ridge()` function has an alpha argument (*lambda*, but with a different name) that is used to tune the model.
- We'll generate an array of alpha values ranging from very big to very small, essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit

```
alphas = 10**np.linspace(10,-2,100)*0.5
print(alphas.size)
```

```
## 100
```

```
print(alphas)
```

```
## [5.00000000e+09 3.78231664e+09 2.86118383e+09 2.16438064e+09
##  1.63727458e+09 1.23853818e+09 9.36908711e+08 7.08737081e+08
##  5.36133611e+08 4.05565415e+08 3.06795364e+08 2.32079442e+08
##  1.75559587e+08 1.32804389e+08 1.00461650e+08 7.59955541e+07
##  5.74878498e+07 4.34874501e+07 3.28966612e+07 2.48851178e+07
##  1.88246790e+07 1.42401793e+07 1.07721735e+07 8.14875417e+06
##  6.16423370e+06 4.66301673e+06 3.52740116e+06 2.66834962e+06
##  2.01850863e+06 1.52692775e+06 1.15506485e+06 8.73764200e+05
```

Python: Ridge Regression

- Associated with each alpha value is a vector of ridge regression coefficients, which we'll store in a matrix coefs. In this case, it is a 19×100 matrix, with 19 rows (one for each predictor) and 100 columns (one for each value of alpha).

```
from sklearn.linear_model import Ridge, RidgeCV
ridge = Ridge()
coefs = []

for a in alphas:
    ridge.set_params(alpha = a)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)
```

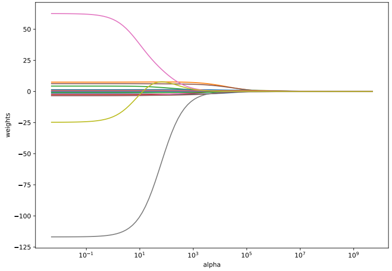
```
## Ridge(alpha=5000000000.0)
## Ridge(alpha=5000000000.0)
## Ridge(alpha=3782316637.773145)
## Ridge(alpha=3782316637.773145)
## Ridge(alpha=2861183829.67511)
## Ridge(alpha=2861183829.67511)
## Ridge(alpha=2164380640.5415306)
## Ridge(alpha=2164380640.5415306)
```

Python: Ridge Regression

```
import matplotlib.pyplot as plt
ax = plt.gca() #get current axes
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.axis('tight')
```

```
## (0.0012559432157547897, 19905358527.674843, -125.81034418305276, 71.5439
```

```
plt.xlabel('alpha')
plt.ylabel('weights')
plt.show()
```



Python: Ridge Regression

- We use cross-validation to choose the tuning parameter λ. We can do this using the cross-validated ridge regression function, `RidgeCV()`. By default, the function performs generalized cross-validation (an efficient form of LOOCV), though this can be changed using the argument cv.

```
ridgecv=RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error')
ridgecv.fit(X, y)
```

```
## RidgeCV(alphas=array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.16438064
##        1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
##        5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
##        1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
##        5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
##        1.88246790e+07, 1.42401793e+0...
##        3.06795364e+00, 2.32079442e+00, 1.75559587e+00, 1.32804389e+00,
##        1.00461650e+00, 7.59955541e-01, 5.74878498e-01, 4.34874501e-01,
##        3.28966612e-01, 2.48851178e-01, 1.88246790e-01, 1.42401793e-01,
##        1.07721735e-01, 8.14875417e-02, 6.16423370e-02, 4.66301673e-02,
##        3.52740116e-02, 2.66834962e-02, 2.01850863e-02, 1.52692775e-02,
##        1.15506485e-02, 8.73764200e-03, 6.60970574e-03, 5.00000000e-03]),
##         scoring='neg_mean_squared_error')
```

```
ridgecv.alpha_
```

```
## 12.38538177995857
```

Python: Ridge Regression

- Now we fit a ridge regression to the full data set using the above λ
  - As expected, none of the coefficients are zero — ridge regression does not perform feature selection!

```
ridge = Ridge()
ridge.set_params(alpha = ridgecv.alpha_)
```

```
## Ridge(alpha=12.38538177995857)
```

```
ridge.fit(X, y)
```

```
## Ridge(alpha=12.38538177995857)
```

```
ridge.coef_
```

```
## array([ -2.03080183,   7.58729214,   4.01413486,  -2.31858126,
##         -0.8747345 ,   6.19350266,  -3.21286106,  -0.17083247,
##          0.10583579,  -0.20532684,   1.49788263,   0.81824231,
##         -0.80925847,   0.28401262,   0.37386431,  -3.19914807,
##         35.21379582, -97.24338426,  -0.40886406])
```

Python: Ridge Regression

- We can define our own cross-validation method to evaluate model

```
from sklearn.model_selection import KFold
#k_fold = KFold(n_splits=10, random_state=1, shuffle=True)
k_fold=10
ridgecv=RidgeCV(alphas = alphas, cv=k_fold, scoring = 'neg_mean_squared_error')
ridgecv.fit(X, y)
```

```
## RidgeCV(alphas=array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.16438064
##        1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
##        5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
##        1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
##        5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
##        1.88246790e+07, 1.42401793e+0...
##        3.06795364e+00, 2.32079442e+00, 1.75559587e+00, 1.32804389e+00,
##        1.00461650e+00, 7.59955541e-01, 5.74878498e-01, 4.34874501e-01,
##        3.28966612e-01, 2.48851178e-01, 1.88246790e-01, 1.42401793e-01,
##        1.07721735e-01, 8.14875417e-02, 6.16423370e-02, 4.66301673e-02,
##        3.52740116e-02, 2.66834962e-02, 2.01850863e-02, 1.52692775e-02,
##        1.15506485e-02, 8.73764200e-03, 6.60970574e-03, 5.00000000e-03]),
##         cv=10, scoring='neg_mean_squared_error')
```

```
ridgecv.alpha_
```

```
## 7599.555414764665
```

Python: Ridge Regression

```
ridge = Ridge()
ridge.set_params(alpha = ridgecv.alpha_)
```

```
## Ridge(alpha=7599.555414764665)
```

```
ridge.fit(X, y)
```

```
## Ridge(alpha=7599.555414764665)
```

```
ridge.coef_
```

```
## array([-1.63383994,  5.23526057, -0.07655521,  0.1782647 ,  0.55534139,
##         4.52856994, -0.07687155, -0.25193904,  0.46930403,  0.16488993,
##         1.23702987,  0.65319635, -0.57663173,  0.29431124,  0.3359052 ,
##        -1.49248217,  0.36453868, -0.95896652,  0.26442831])
```

Python: Ridge Regression

- Lastly, we can use the ridge regression model we chose to make predictions on new observations.

```
new = X.iloc[1,:]+1
# ridge.predict(np.array(new).reshape(1,-1))
ridge.predict([new]) #warning: no feature names
# to remove the warning, make `new` a dataframe
# new2=pd.DataFrame(new).T
# ridge.predict(new2)
```

```
## array([783.07369861])
##
## C:\Users\xzhang2\AppData\Local\Programs\Python\PYTHON~2\lib\site-packages\sklear
##   warnings.warn(
```

Python: Lasso Regression

- In order to fit a lasso model, we'll use the Lasso() function;

```
from sklearn.linear_model import Lasso, LassoCV
lasso = Lasso(max_iter = 10000)
alphas = 10**np.linspace(10,-2,100)*0.5
coefs = []

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(X, y)
    coefs.append(lasso.coef_)
```

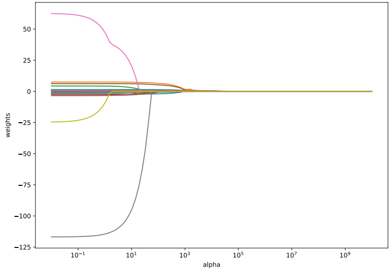
```
## Lasso(alpha=5000000000.0, max_iter=10000)
## Lasso(alpha=5000000000.0, max_iter=10000)
## Lasso(alpha=3782316637.773145, max_iter=10000)
## Lasso(alpha=3782316637.773145, max_iter=10000)
## Lasso(alpha=2861183829.67511, max_iter=10000)
## Lasso(alpha=2861183829.67511, max_iter=10000)
## Lasso(alpha=2164380640.5415306, max_iter=10000)
## Lasso(alpha=2164380640.5415306, max_iter=10000)
## Lasso(alpha=1637274581.438866, max_iter=10000)
## Lasso(alpha=1637274581.438866, max_iter=10000)
```

Python: Lasso Regression

```
import matplotlib.pyplot as plt
ax = plt.gca()
ax.plot(alphas*2, coefs)
ax.set_xscale('log')
plt.axis('tight')
```

```
## (0.0025118864315095794, 39810717055.34969, -125.78998879445157, 71.41109
```

```
plt.xlabel('alpha')
plt.ylabel('weights')
plt.show()
```



Python: Lasso Regression

- Again, we use cross-validation to choose the tuning parameter.
  - We perform 10-fold cross-validation to choose the best alpha

```
lassocv = LassoCV(alphas = None, cv = 10, max_iter = 10000)
lassocv.fit(X, y)
```

```
## LassoCV(cv=10, max_iter=10000)
```

```
lasso = Lasso(max_iter = 10000)
lasso.set_params(alpha=lassocv.alpha_)
```

```
## Lasso(alpha=540.6556677332911, max_iter=10000)
```

```
lasso.fit(X, y)
```

```
## Lasso(alpha=540.6556677332911, max_iter=10000)
```

```
print(lasso.coef_)
```

```
## [-0.          1.49581634  0.          0.          0.          1.13968178
##  -0.         -0.33741746  0.82331967  0.          0.79940936  0.65462067
##  -0.03322761  0.27574744  0.13169212 -0.          0.         -0.
##   0.        ]
```

Python: Lasso Regression

- It can be seen that Lasso regression can perform feature selection!

```
print(pd.Series(lasso.coef_, index=X.columns) )
```

```
## AtBat         -0.000000
## Hits           1.495816
## HmRun          0.000000
## Runs           0.000000
## RBI            0.000000
## Walks          1.139682
## Years         -0.000000
## CAtBat        -0.337417
## CHits          0.823320
## CHmRun         0.000000
## CRuns          0.799409
## CRBI           0.654621
## CWalks        -0.033228
## PutOuts        0.275747
## Assists        0.131692
## Errors        -0.000000
## League_N       0.000000
## Division_W    -0.000000
## NewLeague_N    0.000000
## dtype: float64
```

License