Regularized Regression

Justin Post

Regularization Methods

- Recall the LASSO model (like least squares but a penalty term added)
  - α (>0) is called a tuning parameter

$$\min_{\beta' s} \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}))^2 + \alpha \sum_{j=1}^{p} |\beta_j|$$

- Sets coefficients to 0 as you 'shrink'!

Tuning Parameter

- When choosing the tuning parameter, we are really considering a **family of models**!
- Let's recall an example we did

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from math import sqrt
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression, LassoCV, Lasso
fat_data = pd.read_csv("https://www4.stat.ncsu.edu/~online/datasets/fat.csv")
fat_data.columns
```

```
## Index(['Unnamed: 0', 'brozek', 'siri', 'density', 'age', 'weight', 'height',
##        'adipos', 'free', 'neck', 'chest', 'abdom', 'hip', 'thigh', 'knee',
##        'ankle', 'biceps', 'forearm', 'wrist'],
##       dtype='object')
```

Cleaning and Splitting the Data

- Drop some variables we don't want
- Remove any rows with missing values

```
mod_fat_data = fat_data.drop(["Unnamed: 0", "siri", "density"], axis = 1).dropna()

X_train, X_test, y_train, y_test = train_test_split(
  mod_fat_data.drop("brozek", axis = 1),
  mod_fat_data["brozek"],
  test_size=0.20,
  random_state=41)
```

Scale Data with Regularization

- Usually want to scale the data if using regularization methods
  - Subtract mean, divide by sd
  - Use the training means and sds for test set too!

```
means = X_train.apply(np.mean, axis = 0)
stds = X_train.apply(np.std, axis = 0)
X_train = X_train.apply(lambda x: (x-np.mean(x))/np.std(x), axis = 0)
X_train.head()
```

```
##           age    weight    height  ...    biceps   forearm     wrist
## 120  0.540354  1.015051  1.153840  ...  0.610561  1.235346  1.389566
## 133  0.384191 -0.767741 -0.849386  ...  0.505550  0.307013 -0.955724
## 207  0.149947  0.600867  0.636879  ...  1.590663  1.430785  0.216921
## 49   0.149947 -1.830214 -0.849386  ... -1.874697 -1.403073 -1.488745
## 25  -1.411679 -0.686705  0.378398  ... -0.789584 -0.230442 -0.529308
##
## [5 rows x 15 columns]
```

NC STATE UNIVERSITY

Scale Data with Regularization

- Usually want to scale the data if using regularization methods
  - Subtract mean, divide by sd
  - Use the training means and sds for test set too!

```
#quick function to standardize based off of a supplied mean and std
def my_std_fun(x, means, stds):
    return(x-means)/stds
#loop through the columns and use the function on each
for x in X_test.columns:
    X_test[x] = my_std_fun(X_test[x], means[x], stds[x])
X_test.head()
```

```
##            age    weight    height  ...    biceps   forearm     wrist
## 107  0.540354  0.897999  1.089220  ...  1.030604  0.453592  0.963150
## 143 -1.724004 -0.668697  0.572259  ... -0.579563 -0.719039  0.003713
## 167 -0.787028  1.672344  0.572259  ...  1.765681  2.163679  1.709378
## 29  -1.255516 -0.632681 -0.267804  ... -0.719577 -0.963337 -0.635912
## 30  -1.021272  0.132659  0.959980  ...  0.120510 -0.474741  0.216921
##
## [5 rows x 15 columns]
```

NC STATE UNIVERSITY

Fit a LASSO Model Using CV

```
lasso_mod = LassoCV(cv=5, random_state=0) \
                  .fit(X_train, y_train)
print(lasso_mod.alpha_)
```

```
## 0.0682784472098843
```

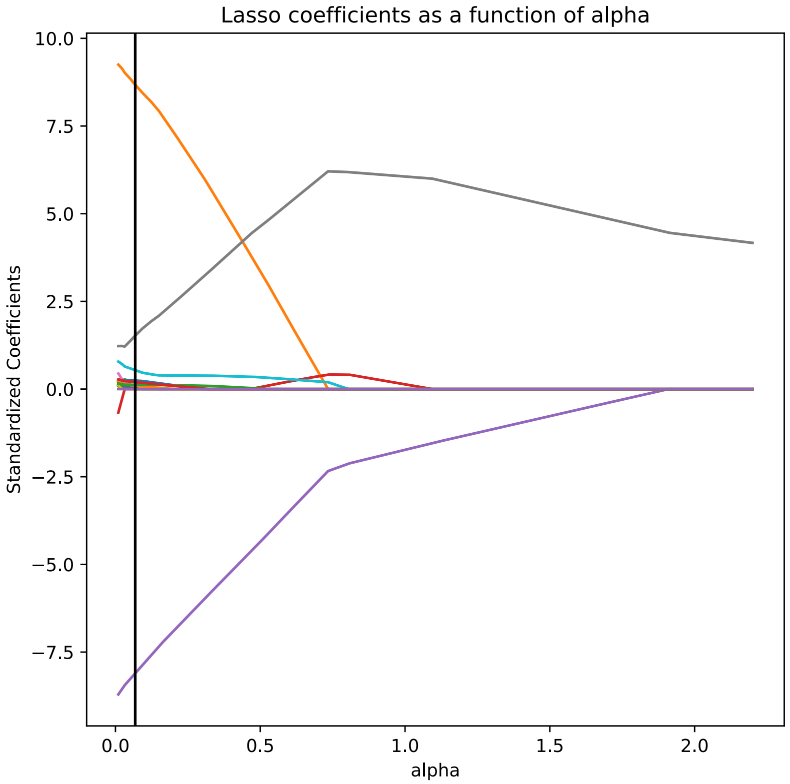
```
print(np.array(list(zip(X_train.columns, lasso_mod.coef_))))
```

```
## [['age' '0.0352062230265082']
##  ['weight' '8.67668517157885']
##  ['height' '0.18524483241596934']
##  ['adipos' '0.0']
##  ['free' '-8.098358879411053']
##  ['neck' '0.0']
##  ['chest' '0.10123124957260958']
##  ['abdom' '1.5227501560784786']
##  ['hip' '0.0']
##  ['thigh' '0.5368436925906938']
##  ['knee' '0.2410290965097115']
##  ['ankle' '0.09972823212694173']
##  ['biceps' '0.12439075979914412']
##  ['forearm' '0.20197553831501175']
##  ['wrist' '0.0']]
```

NC STATE UNIVERSITY

LASSO Fits Visual

```
## (-0.09950000000000003, 2.3095000000000003, -9.605158072913387, 10.14868287610137)
```



Fit 'Best' Model by CV on All Training Data

```
lasso_best = Lasso(lasso_mod.alpha_).fit(X_train,y_train)
```

- Predict on the test set (using the standardized test predictors!)

```
lasso_pred = lasso_best.predict(X_test)
#could compare this to other 'best' models
np.sqrt(mean_squared_error(y_test, lasso_pred))
```

```
## 1.9916053642246037
```

Penalized Regression or Regularized Regression

In linear regression, adding a penalty term to the loss function is called penalized regression or regularized regression.

- L_1 penalty shrinks and does variable selection

$$\min_{\beta's}\sum_{i=1}^{n}(y_i-(\beta_0+\beta_1x_{1i}+\ldots+\beta_px_{pi}))^2+\alpha\sum_{j=1}^{p}|\beta_j|$$

Penalized Regression or Regularized Regression

In linear regression, adding a penalty term to the loss function is called penalized regression or regularized regression.

- L_1 penalty shrinks and does variable selection

$$\min_{\beta' s} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}))^2 + \alpha \sum_{j=1}^{p} |\beta_j|$$

- L_2 penalty shrinks coefficients (works well for multicollinearity)

$$\min_{\beta' s} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Penalized Regression or Regularized Regression

In linear regression, adding a penalty term to the loss function is called penalized regression or regularized regression.

- L_1 penalty shrinks and does variable selection

$$\min_{\beta's} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}))^2 + \alpha \sum_{j=1}^{p} |\beta_j|$$

- L_2 penalty shrinks coefficients (works well for multicollinearity)

$$\min_{\beta's} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- L_1 and L_2 penalties combine the approaches

$$\min_{\beta's} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}))^2 + \alpha \sum_{j=1}^{p} |\beta_j| + \lambda \sum_{j=1}^{p} \beta_j^2$$

Penalized Regression or Regularized Regression

- For MLR, these can be done via
  - `sklearn.linear_model.Lasso`
  - `sklearn.linear_model.Ridge`
  - `sklearn.linear_model.ElasticNet`

Penalized Regression or Regularized Regression

- For MLR, these can be done via
  - `sklearn.linear_model.Lasso`
  - `sklearn.linear_model.Ridge`
  - `sklearn.linear_model.ElasticNet`
- `sklearn.linear_model.*CV` to easily use CV!
- Tuning parameters for Elastic Net:

$$\min_{\beta' s} \frac{1}{2n}\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}))^2$$

$$+\alpha * L1\_ratio \sum_{j=1}^{p} |\beta_j| + 0.5 * \alpha(1 - L1\_ratio) \sum_{j=1}^{p} \beta_j^2$$

Elastic Net

```
from sklearn.linear_model import ElasticNetCV
regr = ElasticNetCV(cv=5,
                    random_state=0,
                    l1_ratio = [0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.96, 0.98, 0.99, 1],
                    n_alphas = 50)
regr.fit(X_train, y_train)
```

```
print(regr.alpha_)
```

```
## 0.0682784472098843
```

```
print(regr.l1_ratio_)
```

```
## 1.0
```

Elastic Net

- Refit on full training data with best tuning parameters

```
from sklearn.linear_model import ElasticNet
en = ElasticNet(alpha = regr.alpha_, l1_ratio = regr.l1_ratio_)
en.fit(X_train, y_train)
```

```
print(np.array(list(zip(X_train.columns, en.coef_))))
```

```
## [['age' '0.0352062230265082']
##  ['weight' '8.67668517157885']
##  ['height' '0.18524483241596934']
##  ['adipos' '0.0']
##  ['free' '-8.098358879411053']
##  ['neck' '0.0']
##  ['chest' '0.10123124957260958']
##  ['abdom' '1.5227501560784786']
##  ['hip' '0.0']
##  ['thigh' '0.5368436925906938']
##  ['knee' '0.2410290965097115']
##  ['ankle' '0.09972823212694173']
##  ['biceps' '0.12439075979914412']
##  ['forearm' '0.20197553831501175']
##  ['wrist' '0.0']]
```

Compare on Test Set

```
lasso_pred = lasso_best.predict(X_test)
en_pred = en.predict(X_test)
print([np.sqrt(mean_squared_error(y_test, lasso_pred)),
       np.sqrt(mean_squared_error(y_test, en_pred))])
```

```
## [1.9916053642246037, 1.9916053642246037]
```

Regularized Logistic Regression

- Same ideas here!
- `sklearn.linear_model.LogisticRegression` can do all three penalized methods mentioned
    - `penalty` = 'l1', 'l2', 'elasticnet', or `none`
    - `default='l2'`! (`C` is regularization parameter = 1 by default)
    - For elastic net, `solver = 'saga'` and specify `l1_ratio`

Regularized Logistic Regression

- Same ideas here!
- `sklearn.linear_model.LogisticRegression` can do all three penalized methods mentioned
    - `penalty` = 'l1', 'l2', 'elasticnet', or `none`
    - `default='l2'`! (`C` is regularization parameter = 1 by default)
    - For elastic net, `solver = 'saga'` and specify `l1_ratio`
- `sklearn.linear_model.LogisticRegressionCV` for CV!

Quick Example

- Make a binary version of response

```
y_train2 = y_train < 25
y_test2 = y_test < 25
```

Quick Example

- Make a binary version of response

```
y_train2 = y_train < 25
y_test2 = y_test < 25
```

- Fit `L2` regularized logistic regression

```
from sklearn.linear_model import LogisticRegressionCV
log_reg_cv = LogisticRegressionCV(cv = 5,
                              solver = "newton-cg",
                              penalty = "l2",
                              Cs = 250,
                              scoring = "neg_log_loss",
                              random_state = 10)
```

```
log_reg_cv.fit(X_train, y_train2)
```

Results

- Optimal regularization value (smaller means more regularized)

```
log_reg_cv.C_
```

```
## array([1.29553694])
```

- Fit optimal model

```
from sklearn.linear_model import LogisticRegression
log_reg_best_cv = LogisticRegression(solver = "newton-cg",
                             penalty = "l2",
                             C = log_reg_cv.C_[0],
                             random_state = 5)
```

```
log_reg_best_cv.fit(X_train, y_train2)
```

Compare Coefficients

- Compare non-regularize model with regularized:

```
log_reg_full = LogisticRegression(solver = "newton-cg", penalty = "none", random_state = 0)
log_reg_full.fit(X_train, y_train2)
```

Compare Coefficients

- Compare non-regularize model with regularized:

```
log_reg_full = LogisticRegression(solver = "newton-cg", penalty = "none", random_state = 0)
log_reg_full.fit(X_train, y_train2)
```

```
for i in range(log_reg_full.coef_.shape[1]):
  print(X_train.columns[i], log_reg_full.coef_[:,i], log_reg_best_cv.coef_[:,i])
```

```
## age [-2.85330696] [-0.67147741]
## weight [-122.51657579] [-1.4770971]
## height [-6.48047944] [-0.10251239]
## adipos [-1.44034225] [-0.30237501]
## free [114.5091415] [3.52712708]
## neck [0.23516711] [-0.33668417]
## chest [-9.31213962] [-1.12477439]
## abdom [-14.39030691] [-1.61212997]
## hip [16.30752518] [0.25746399]
## thigh [-3.00002705] [-0.55292862]
## knee [-7.14568787] [-0.32024622]
## ankle [-0.62655066] [-0.35822733]
## biceps [-9.15749311] [-0.02451164]
## forearm [2.70724672] [-0.48905071]
## wrist [5.78592559] [0.45770954]
```

Compare on Test Data

- Which model generalizes better?

```
cv_proba_preds = log_reg_best_cv.predict_proba(X_test)
full_proba_preds = log_reg_full.predict_proba(X_test)

from sklearn.metrics import log_loss, accuracy_score
log_loss(y_test2, cv_proba_preds)
```

```
## 0.1681322951793145
```

```
log_loss(y_test2, full_proba_preds)
```

```
## 0.5433586256880958
```

```
log_loss(y_test2, np.array([[0,1] for _ in range(len(y_test2.values))]))
```

```
## 8.126770916449573
```

Compare on Test Data

- Which model generalizes better?

```
cv_preds = log_reg_best_cv.predict(X_test)
full_preds = log_reg_full.predict(X_test)

from sklearn.metrics import log_loss, accuracy_score
accuracy_score(y_test2, cv_preds)
```

```
## 0.9411764705882353
```

```
accuracy_score(y_test2, full_preds)
```

```
## 0.9607843137254902
```

```
accuracy_score(y_test2, np.array([1 for _ in range(len(y_test2.values))]))
```

```
## 0.7647058823529411
```

Complicated Process

- Process often pretty involved
    - Split data
    - Create dummy variables, interaction terms, standardize data, etc.
    - Fit a model, often with CV
    - Choose best model
    - Predict on test set (using appropriate transformations from training set!)

Complicated Process

- Process often pretty involved

    - Split data
    - Create dummy variables, interaction terms, standardize data, etc.
    - Fit a model, often with CV
    - Choose best model
    - Predict on test set (using appropriate transformations from training set!)

- If we were to fit a LASSO model, a Ridge Regression model, and an Elastic Net model, only the 'fit' part really has to change!

- Future: Put process into a **pipeline** for ease!

Recap

- Regularization can improve prediction and do variable selection at the same time
- Implemented for both MLR type models and logistic regression type models