towardsdatascience.com

Ridge and Lasso Regression: L1 and L2 Regularization

Saptashwa Bhattacharyya

13-16 minutes

Complete Guide Using Scikit-Learn

Moving on from a very important unsupervised learning technique that I have discussed last week, today we will dig deep in to supervised learning through linear regression, specifically two special linear regression model — Lasso and Ridge regression.

As I'm using the term linear, first let's clarify that linear models are one of the simplest way to predict output using a linear function of input features.

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + ... + w[n] \times x[n] + b \qquad (1.1)$$

Linear model with n features for output prediction

In the equation (1.1) above, we have shown the linear model based on the n number of features. Considering only a single feature as you probably already have understood that *w[0]* will be slope and *b* will represent intercept. Linear regression looks for optimizing *w* and *b* such that it minimizes the cost function. The cost function can be written as

$$\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j \times x_{ij}\right)^2 \qquad (1.2)$$

Cost function for simple linear model

In the equation above I have assumed the data-set has M instances and p features. Once we use linear regression on a data-set divided in to training and test set, calculating the scores on training and test set can give us a rough idea about whether the model is suffering from over-fitting or under-fitting. The chosen linear model can be just right also, if you're lucky enough! If we have very few features on a data-set and the score is poor for both training and test set then it's a problem of under-fitting. On the other hand if we have large number of features and test score is relatively poor than the training score then it's the problem of over-generalization or over-fitting. **Ridge and Lasso regression are some of the simple techniques to reduce model complexity and prevent over-fitting which may result from simple linear regression**.

Ridge Regression : In ridge regression, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients.

$$\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j \times x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} w_j^2 \qquad (1.3)$$

Cost function for ridge regression

This is equivalent to saying minimizing the cost function in equation 1.2 under the condition as below

$$\text{For some } c > 0,\ \sum_{j=0}^{p} w_j^2 < c$$

Supplement 1: Constrain on Ridge regression coefficients

So ridge regression puts constraint on the coefficients *(w).* The penalty term (lambda) regularizes the coefficients such that if the coefficients take large values the optimization function is penalized. So, **ridge regression shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity.** Going back to eq. 1.3 one can see that when λ → 0 , the cost function becomes similar to the linear regression cost function (eq. 1.2). So *lower the constraint (low λ) on the features, the model will resemble linear regression model.* Let's see an example using Boston house data and below is the code I used to depict linear regression as a limiting case of Ridge regression-

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams.update({'font.size': 12})from
sklearn.datasets import load_boston
from sklearn.cross_validation import
train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import
Ridgeboston=load_boston()
boston_df=pd.DataFrame(boston.data,columns=boston.feature_names)
#print boston_df.info()# add another column that
contains the house prices which in scikit learn
datasets are considered as target
boston_df['Price']=boston.target
#print
boston_df.head(3)newX=boston_df.drop('Price',axis=1)
print newX[0:3] # check
newY=boston_df['Price']#print type(newY)# pandas
core
frameX_train,X_test,y_train,y_test=train_test_split(newX,newY,test_size=0.3,random_sta
print len(X_test), len(y_test)lr =
LinearRegression()
lr.fit(X_train, y_train)rr = Ridge(alpha=0.01) #
higher the alpha value, more restriction on the
coefficients; low alpha > more generalization,
# in this case linear and ridge regression
resemblesrr.fit(X_train, y_train)rr100 =
Ridge(alpha=100) #  comparison with alpha value
rr100.fit(X_train,
y_train)train_score=lr.score(X_train, y_train)
test_score=lr.score(X_test,
y_test)Ridge_train_score =
rr.score(X_train,y_train)
Ridge_test_score = rr.score(X_test,
y_test)Ridge_train_score100 =
rr100.score(X_train,y_train)
Ridge_test_score100 = rr100.score(X_test,
y_test)plt.plot(rr.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red
$\alpha = 0.01$',zorder=7)
plt.plot(rr100.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',l
$\alpha = 100$')
plt.plot(lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='green',lab
Regression')plt.xlabel('Coefficient
Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
plt.show()
```

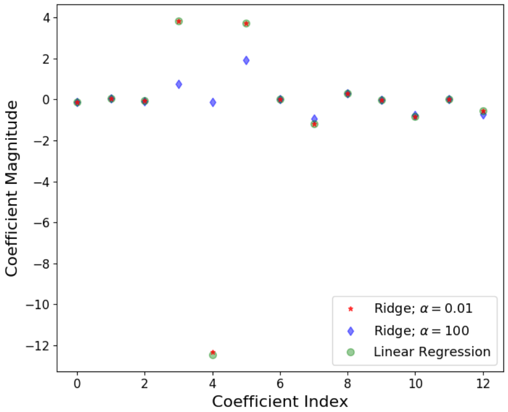

Figure 1: Ridge regression for different values of alpha is plotted to show linear regression as limiting case of ridge regression. Source: Author.

Let's understand the figure above. In X axis we plot the coefficient index and, for Boston data there are 13 features (for Python 0th index refers to 1st feature). For low value of α (0.01), when the coefficients are less restricted, the magnitudes of the coefficients are almost same as of linear regression. For higher value of α (100), we see that for coefficient indices 3,4,5 the magnitudes are considerably less compared to linear regression case. This is an example of *shrinking* coefficient magnitude using Ridge regression.

Lasso Regression : The cost function for Lasso (least absolute shrinkage and selection operator) regression can be written as

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} |w_j| \qquad (1.4)$$

Cost function for Lasso regression

$$\text{For some } t > 0,\ \sum_{j=0}^{p} |w_j| < \ t$$

Supplement 2: Lasso regression coefficients; subject to similar constrain as Ridge, shown before.

Just like Ridge regression cost function, for lambda =0, the equation above reduces to equation 1.2. *The only difference is instead of taking the square of the coefficients, magnitudes are taken into account.* This type of regularization (L1) can lead to zero coefficients i.e. some of the features are completely neglected for the evaluation of output. **So Lasso regression not only helps in reducing over-fitting but it can help us in feature selection.** Just like Ridge regression the regularization parameter (lambda) can be controlled and we will see the effect below using cancer data set in `sklearn`. Reason I am using cancer data instead of Boston house data, that I have used before, is, cancer data-set have 30 features compared to only 13 features of Boston house data. So feature selection using Lasso regression can be depicted well by changing the regularization parameter.

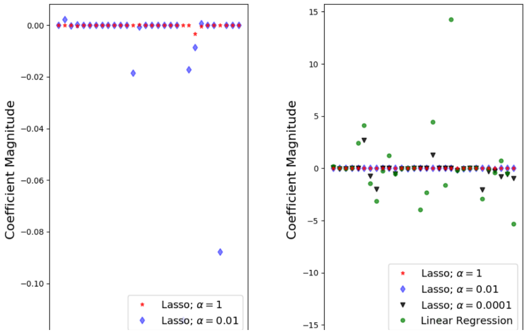

Figure 2: Lasso regression and feature selection dependence on the regularization parameter value. Source: Author.

The code I used to make these plots is as below

```
import math
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np# difference of lasso and ridge
regression is that some of the coefficients can be
zero i.e. some of the features are
# completely neglectedfrom sklearn.linear_model
import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_breast_cancer
from sklearn.cross_validation import
train_test_splitcancer = load_breast_cancer()
#print cancer.keys()cancer_df =
pd.DataFrame(cancer.data,
columns=cancer.feature_names)#print
cancer_df.head(3)X = cancer.data
Y =
cancer.targetX_train,X_test,y_train,y_test=train_test_split(X,Y,
test_size=0.3, random_state=31)lasso = Lasso()
lasso.fit(X_train,y_train)
train_score=lasso.score(X_train,y_train)
test_score=lasso.score(X_test,y_test)
coeff_used = np.sum(lasso.coef_!=0)print "training
score:", train_score
print "test score: ", test_score
print "number of features used: ",
coeff_usedlasso001 = Lasso(alpha=0.01,
max_iter=10e5)
lasso001.fit(X_train,y_train)train_score001=lasso001.score(X_train,y_train)
test_score001=lasso001.score(X_test,y_test)
coeff_used001 = np.sum(lasso001.coef_!=0)print
"training score for alpha=0.01:", train_score001
print "test score for alpha =0.01: ",
test_score001
print "number of features used: for alpha =0.01:",
coeff_used001lasso00001 = Lasso(alpha=0.0001,
max_iter=10e5)
lasso00001.fit(X_train,y_train)train_score00001=lasso00001.score(X_train,y_train)
test_score00001=lasso00001.score(X_test,y_test)
coeff_used00001 = np.sum(lasso00001.coef_!=0)print
"training score for alpha=0.0001:",
train_score00001
print "test score for alpha =0.0001: ",
test_score00001
print "number of features used: for alpha
=0.0001:", coeff_used00001lr = LinearRegression()
lr.fit(X_train,y_train)
lr_train_score=lr.score(X_train,y_train)
lr_test_score=lr.score(X_test,y_test)print "LR
training score:", lr_train_score
print "LR test score: ",
lr_test_scoreplt.subplot(1,2,1)
plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',la
$\alpha = 1$',zorder=7) # alpha here is for
transparency
```

```
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue
$\alpha = 0.01$') # alpha here is for transparency
plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
```

```
plt.subplot(1,2,2)plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersiz
$\alpha = 1$',zorder=7) # alpha here is for
transparency
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue
$\alpha = 0.01$') # alpha here is for transparency
plt.plot(lasso00001.coef_,alpha=0.8,linestyle='none',marker='v',markersize=6,color='bl
$\alpha = 0.00001$') # alpha here is for
transparency
plt.plot(lr.coef_,alpha=0.7,linestyle='none',marker='o',markersize=5,color='green',lab
Regression',zorder=2)plt.xlabel('Coefficient
Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
plt.tight_layout()
plt.show()
#output training score: 0.5600974529893081
test score:  0.5832244618818156
number of features used:  4training score for
alpha=0.01: 0.7037865778498829
test score for alpha =0.01:  0.664183157772623
number of features used: for alpha =0.01:
10training score for alpha=0.0001:
0.7754092006936697
test score for alpha =0.0001:  0.7318608210757904
number of features used: for alpha =0.0001: 22LR
training score: 0.7842206194055068
LR test score:  0.7329325010888681
```

Let's understand the plot and the code in a short summary.

- The default value of regularization parameter in Lasso regression (given by α) is 1.
- With this, out of 30 features in cancer data-set, only 4 features are used (non zero value of the coefficient).
- Both training and test score (with only 4 features) are low; conclude that the model is under-fitting the cancer data-set.
- Reduce this under-fitting by reducing alpha and increasing number of iterations. Now α = 0.01, non-zero features =10, training and test score increases.
- Comparison of coefficient magnitude for two different values of alpha are shown in the left panel of figure 2. For alpha =1, we can see most of the coefficients are zero or nearly zero, which is not the case for alpha=0.01.
- Further reduce α =0.0001, non-zero features = 22. Training and test scores are similar to basic linear regression case.
- In the right panel of figure, for α = 0.0001, coefficients for Lasso regression and linear regression show close resemblance.

How Lasso Regularization Leads to Feature Selection?

So far we have gone through the basics of Ridge and Lasso regression and seen some examples to understand the applications. Now, I will try to explain why the Lasso regression can result in feature selection and Ridge regression only reduces the coefficients close to zero, but not zero. An illustrative figure below will help us to understand better, where we will assume a hypothetical data-set with only two features. Using the constrain for the coefficients of Ridge and Lasso regression (as shown above in the supplements 1 and 2), we can plot the figure below

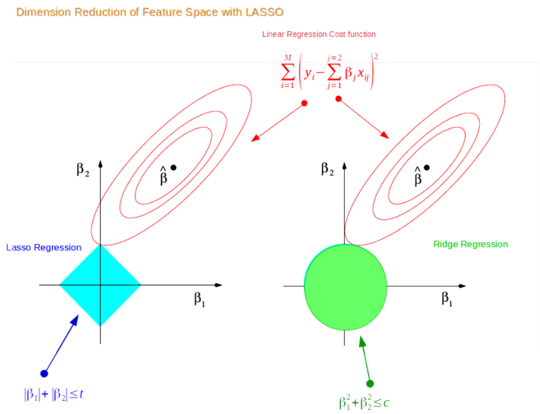
Dimension Reduction of Feature Space with LASSO
Linear Regression Cost function
$\sum_{i=1}^{M}\left(y_i-\sum_{j=1}^{j=2}\beta_j x_{ij}\right)^2$
β_2
$\hat{\beta}$
Lasso Regression
β_1
$|\beta_1|+|\beta_2|\leq t$
β_2
$\hat{\beta}$
Ridge Regression
β_1
$\beta_1^2+\beta_2^2\leq c$

Figure 3: Why LASSO can reduce dimension of feature space? Example on 2D feature space. Modified from the plot used in ‘The Elements of Statistical Learning’ by Author.

For a two dimensional feature space, the constraint regions (see supplement 1 and 2) are plotted for Lasso and Ridge regression with cyan and green colours. The elliptical contours are the cost function of linear regression (eq. 1.2). Now if we have relaxed conditions on the coefficients, then the constrained regions can get bigger and eventually they will hit the centre of the ellipse. This is the case when Ridge and Lasso regression resembles linear regression results. Otherwise, **both methods determine coefficients by finding the first point where the elliptical contours hit the region of constraints. The diamond (Lasso) has corners on the axes, unlike the disk, and whenever the elliptical region hits such point, one of the features completely vanishes!** For higher dimensional feature space there can be many solutions on the axis with Lasso regression and thus we get only the important features selected.

Finally to end this meditation, let’s summarize what we have learnt so far

1. Cost function of Ridge and Lasso regression and importance of regularization term.
2. Went through some examples using simple data-sets to understand Linear regression as a limiting case for both Lasso and Ridge regression.
3. Understood why Lasso regression can lead to feature selection whereas Ridge can only shrink coefficients close to zero.

For further reading I suggest “The element of statistical learning”; J. Friedman et.al., Springer, pages- 79-91, 2008. Examples shown here to demonstrate regularization using L1 and L2 are influenced from the fantastic Machine Learning with Python book by Andreas Muller.

Hope you have enjoyed the post and stay happy ! Cheers !

P.S: Please see the comment made by Akanksha Rawat for a critical view on standardizing the variables before applying Ridge regression algorithm.