Modern Optimization Methods (Part II) HW2

November 20, 2020

1 A Lasso-linear model via k-fold cross validation for the drug Erlotinib

R09946006 | HO Ching Ru

In p. 748 of Irorio et al. (2016), they built linear (regression with elastic net) and nonlinear predictive models (random forest). Please build a linear or nonlinear model via k-fold cross validation for the drug Erlotinib, whose gene expression data and drug response will be sent; choose any k for your preference, but explain why.

1.1 Recap: Objective Function of Linear Regression

The target of a Linear Regression model is try to find the parameters to minimize the SSE (Sum of Squared Error), thus the objective function is:

$$\min\{SSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2\}$$

1.2 Ridge Regression

Add the penalty parameter $\sum_{j=1}^{p}\beta_j^2$ term into objective function:

$$\min\{SSE + \lambda\sum_{j=1}^{p}\beta_j^2\} = \min\{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\}$$

1.3 Lasso Regression

Similar to Ridge Regression, but in Lasso Regression we add the penalty parameter $\sum_{j=1}^{p}|\beta_j|$ term into objective function:

$$\min\{SSE + \lambda\sum_{j=1}^{p}|\beta_j|\} = \min\{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\}$$

1.4 Elastic Net

The moethod which is used in Irorio et al. (2016). Combine Ridge Regression and Lasso Regression together:

$$\min\{SSE + \lambda_1 \sum_{j=1}^{p} \beta_j^2 + \lambda_2 \sum_{j=1}^{p} |\beta_j|\} = \min\{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{p} \beta_j^2 + \lambda_2 \sum_{j=1}^{p} |\beta_j|\}$$

2 Import dataset and pretreatment

[39]:
```
import pandas as pd
import numpy as np

# read the csv files
train_dataset = pd.read_csv('Erlotinib_data_of_GDSC.csv', sep=',')
test_data = pd.read_csv('Erlotinib_data_of_Clinical_trials.csv', sep=',')
```

2.1 Check training dataset (Erlotinib data of GDSC):

- Row data: cell lines
- Column data: genes

First column is **IC50**, when **IC50**<2 means **sensitive**(1), otherwise **IC50**>2 means **resistant**(0).

[40]:
```
print("dimension of training data: ", np.shape(train_dataset))
train_dataset.head()
```

```
dimension of training data:  (370, 16384)
```

[40]:

	Unnamed: 0	IC50	TSPAN6	TNMD	DPM1	SCYL3	C1orf112	\
0	NCI-H1648	0.043608	1.556156	-0.375902	0.204978	-0.235733	-0.880868	
1	TE-5	1.740005	1.428097	0.658912	0.814844	-0.311046	-0.149503	
2	BB30-HNC	0.154884	0.158151	-0.161001	0.684983	-0.583560	-1.220908	
3	LB2241-RCC	0.965535	1.155232	-0.662094	0.112572	-0.936510	-1.274317	
4	LB996-RCC	0.163644	0.902276	-0.450095	-0.367628	-0.768532	-0.907330	

	FGR	CFH	FUCA2	…	LCAT	PTPRCAP	ANXA2P1	KRTAP4.12	\
0	-0.449110	1.030131	0.911762	…	0.237377	-0.576731	-0.494136	-1.055604	
1	-0.334791	1.875891	1.063786	…	0.686326	-0.736555	-0.978688	-0.615822	
2	-0.480081	-0.249721	-0.062494	…	0.504674	-0.642266	-0.507873	-0.661958	
3	-0.467822	-0.351553	1.027963	…	-0.057632	-0.723391	-0.961332	-1.611498	
4	-0.329345	1.965535	1.280312	…	0.117100	-0.407520	-0.162624	-0.705393	

	KRTAP2.4	GPC2	SIPA1	SRA1	ZBTB9	NDUFS3
0	-1.591790	-0.873141	1.307980	0.953857	0.179825	-1.257159
1	0.588803	-0.611486	0.415982	-1.011809	-0.499291	-1.086598
2	-0.956032	-1.015273	-0.335297	1.016546	0.137157	-0.743470
3	0.224001	-0.792543	-0.908972	-0.262708	-0.664085	-1.822067
4	-1.175140	-0.969499	0.085786	-0.094959	-0.879946	-0.619576

[5 rows x 16384 columns]

2.2 Check testing dataset (Erlotinib data of Clinical trials)

- Row data: patients
- Column data: genes

First column is **OS**, when **OS**= 1 means **sensitive**(1), otherwise **OS**= 0 means **resistant**(0).

[41]:
```
print("dimension of testing data: ", np.shape(test_data))
test_data.head()
```

```
dimension of testing data:  (25, 16384)
```

[41]:

	Unnamed: 0	OS	TSPAN6	TNMD	…	LCAT	PTPRCAP	ANXA2P1	KRTAP4.12	KRTAP2.4	GPC2	SIPA1	SRA1	ZBTB9	NDUFS3
0	GSM677318.CEL	1	0.216298	-0.355524	…	0.545224	-1.315428	-0.097292	-0.257114	0.145398	-0.160226	-0.192911	-0.268092	-1.264167	-0.492084
1	GSM677321.CEL	1	-0.321020	-0.260544	…	-0.318842	-0.719506	-0.646417	-0.380182	-0.722955	0.150588	0.125850	-0.751297	-0.085955	-1.102714
2	GSM677326.CEL	1	1.782405	-0.195986	…	-0.568127	-0.342183	-0.520109	0.387100	0.786042	0.153249	-0.825714	-0.574070	0.165233	-0.689874
3	GSM677327.CEL	0	1.245684	3.664188	…	-0.442494	-0.792073	0.313381	-1.137882	-0.848852	-1.366039	-0.749765	1.049447	0.015924	0.293144
4	GSM789976_Human_Gene_1.0_ST_525_LM_116_05_13_0…	0	-0.646783	-0.527883	…	1.463322	-0.078694	0.936124	-1.102334	-0.138195	-1.009814	-0.192911	-0.898116	0.044284	0.303423

[5 rows x 16384 columns]

2.3 Pretreatment

Separate the dataset into data (X, here are gene data) and target (y, here are sensitive and resistant).

[44]:
```
train_X = train_dataset.iloc[:,2:] # X
train_y = train_dataset.iloc[:,1]  # Y, IC50
```

```
print("dimension of X in training set:", np.shape(train_X))
print("dimension of y in training set:", np.shape(train_y))
```

```
dimension of X in training set: (370, 16382)
dimension of y in training set: (370,)
```

[45]:
```
test_X = test_data.iloc[:,2:] # X
test_y = test_data.iloc[:,1]  # Y, OS
print("dimension of X in testing set:", np.shape(test_X))
print("dimension of y in testing set:", np.shape(test_y))
```

```
dimension of X in testing set: (25, 16382)
dimension of y in testing set: (25,)
```

3 Create a regression model

Choose $\alpha = 1.0$ in Lasso Regression. Constant that multiplies the L_1 term. $\alpha = 0$ is equivalent to an ordinary least square, solved by the LinearRegression object.

3.1 Feature Selection: Lasso Method

[55]:
```
from numpy import mean
from numpy import std
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import Lasso

alphavalue = 1.0

model = Lasso(alpha = alphavalue)
model.fit(train_X, train_y)
type(model)
```

[55]:
```
sklearn.linear_model._coordinate_descent.Lasso
```

3.2 Apply k-fold cross validation

In here, I choose k as 10.

[ ]:
```
kfoldvalue = 10
cv = RepeatedKFold(n_splits = kfoldvalue, n_repeats = 3, random_state = 1)
```

3.3 Return a score about training dataset

[53]:
```
scores = cross_val_score(model, train_X, train_y,␣
 ↪scoring='neg_mean_absolute_error', cv = cv, n_jobs = -1)
scores = absolute(scores)
print('Mean MAE: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Mean MAE: 11.381 (1.544)
```

[ ]:
```

```

[31]:
```
row = list(test_X.iloc[0])
```

[32]:
```
yhat = model.predict([row])
```

[33]:
```
print('Predicted: %.3f' % yhat)
```

```
Predicted: 16.735
```

[ ]:
```

```