Lasso Regression

Lasso Regression with Python

Jan Kirenz

Last updated on Dec 27, 2021 · 12 min read · Python, Statistics, Regression

1 Lasso regression in Python

1.1 Basics

This tutorial is mainly based on the excellent book "An Introduction to Statistical Learning" from James et al. (2021), the scikit-learn documentation about regressors with variable selection as well as Python code provided by Jordi Warmenhoven in this GitHub repository.

Lasso regression relies upon the linear regression model but additionaly performs a so called `L1 regularization`, which is a process of introducing additional information in order to prevent overfitting. As a consequence, we can fit a model containing all possible predictors and use lasso to perform variable selection by using a technique that regularizes the coefficient estimates (it shrinks the coefficient estimates towards zero). In particular, the minimization objective does not only include the residual sum of squares (RSS) - like in the OLS regression setting - but also the sum of the absolute value of coefficients.

The residual sum of squares (RSS) is calculated as follows:

$$RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

This formula can be stated as:

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

- n represents the number of observations.
- p denotes the number of variables that are available in the dataset.
- x_{ij} represents the value of the jth variable for the ith observation, where i = 1, 2, . . ., n and j = 1, 2, . . . , p.

In the lasso regression, the minimization objective becomes:

$$\sum_{i=1}^{n}\left(y_i - (\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij})\right)^2 + \alpha\sum_{j=1}^{p}|\beta_j|$$

which equals:

$$RSS + \alpha\sum_{j=1}^{p}|\beta_j|$$

α (alpha) can take various values:

- α = 0: Same coefficients as least squares linear regression
- α = ∞: All coefficients are zero
- 0 < α < ∞: coefficients are between 0 and that of least squares linear regression

Lasso regression's advantage over least squares linear regression is rooted in the bias-variance trade-off. As α increases, the flexibility of the lasso regression fit decreases, leading to decreased variance but increased bias. This procedure is more restrictive in estimating the coefficients and - depending on your value of α - may set a number of them to exactly zero. This means in the final model the response variable will only be related to a small subset of the predictors—namely, those with nonzero coeffcient estimates. Therefore, selecting a good value of α is critical.

1.2 Data

We illustrate the use of lasso regression on a data frame called "Hitters" with 20 variables and 322 observations of major league players (see this documentation for more information about the data). We want to predict a baseball player's salary on the basis of various statistics associated with performance in the previous year.

1.2.1 Import

```
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/kirenz/datasets/master/Hitters.csv")
```

```
df
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks
0	293	66	1	30	29	14	1	293	66	1	30	29	14
1	315	81	7	24	38	39	14	3449	835	69	321	414	375
2	479	130	18	66	72	76	3	1624	457	63	224	266	263
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354
4	321	87	10	39	42	30	2	396	101	12	48	46	33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
317	497	127	7	65	48	37	5	2703	806	32	379	311	138
318	492	136	5	76	50	94	12	5511	1511	39	897	451	875
319	475	126	3	61	43	52	6	1700	433	7	217	93	146
320	573	144	9	85	60	78	8	3198	857	97	470	420	332
321	631	170	9	77	44	31	11	4908	1457	30	775	357	249

322 rows × 20 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 322 entries, 0 to 321
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   AtBat      322 non-null    int64  
 1   Hits       322 non-null    int64  
 2   HmRun      322 non-null    int64  
 3   Runs       322 non-null    int64  
 4   RBI        322 non-null    int64  
 5   Walks      322 non-null    int64  
 6   Years      322 non-null    int64  
 7   CAtBat     322 non-null    int64  
 8   CHits      322 non-null    int64  
 9   CHmRun     322 non-null    int64  
 10  CRuns      322 non-null    int64  
 11  CRBI       322 non-null    int64  
 12  CWalks     322 non-null    int64  
 13  League     322 non-null    object 
 14  Division   322 non-null    object 
 15  PutOuts    322 non-null    int64  
 16  Assists    322 non-null    int64  
 17  Errors     322 non-null    int64  
 18  Salary     263 non-null    float64
 19  NewLeague  322 non-null    object 
dtypes: float64(1), int64(16), object(3)
memory usage: 50.4+ KB
```

1.2.2 Missing values

Note that the salary is missing for some of the players:

```
print(df.isnull().sum())
```

```
AtBat         0
Hits          0
HmRun         0
Runs          0
RBI           0
Walks         0
Years         0
CAtBat        0
CHits         0
CHmRun        0
CRuns         0
CRBI          0
CWalks        0
League        0
Division      0
PutOuts       0
Assists       0
Errors        0
Salary       59
NewLeague     0
dtype: int64
```

We simply drop the missing cases:

```
# drop missing cases
df = df.dropna()
```

1.2.3 Create labels and features

Since we will use the lasso algorithm from scikit learn, we need to encode our categorical features as one-hot numeric features (dummy variables):

```
dummies = pd.get_dummies(df[['League', 'Division','NewLeague']])
```

```
dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 263 entries, 1 to 321
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   League_A     263 non-null    uint8
 1   League_N     263 non-null    uint8
 2   Division_E   263 non-null    uint8
 3   Division_W   263 non-null    uint8
 4   NewLeague_A  263 non-null    uint8
 5   NewLeague_N  263 non-null    uint8
dtypes: uint8(6)
memory usage: 3.6 KB
```

```
print(dummies.head())
```

```
   League_A  League_N  Division_E  Division_W  NewLeague_A  NewLeague_N
1         0         1           0           1            0            1
2         1         0           0           1            1            0
3         0         1           1           0            0            1
4         0         1           1           0            0            1
5         1         0           0           1            1            0
```

Next, we create our label y:

```
y = df['Salary']
```

We drop the column with the outcome variable (Salary), and categorical columns for which we already created dummy variables:

```
X_numerical = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1).astype('float64
```

Make a list of all numerical features (we need them later):

```
list_numerical = X_numerical.columns
list_numerical
```

```
Index(['AtBat', 'Hits', 'HmRun', 'Runs', 'RBI', 'Walks', 'Years', 'CAtBat',
       'CHits', 'CHmRun', 'CRuns', 'CRBI', 'CWalks', 'PutOuts', 'Assists',
       'Errors'],
      dtype='object')
```

```
# Create all features
X = pd.concat([X_numerical, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 263 entries, 1 to 321
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   AtBat        263 non-null    float64
 1   Hits         263 non-null    float64
 2   HmRun        263 non-null    float64
 3   Runs         263 non-null    float64
 4   RBI          263 non-null    float64
 5   Walks        263 non-null    float64
 6   Years        263 non-null    float64
 7   CAtBat       263 non-null    float64
 8   CHits        263 non-null    float64
 9   CHmRun       263 non-null    float64
 10  CRuns        263 non-null    float64
 11  CRBI         263 non-null    float64
 12  CWalks       263 non-null    float64
 13  PutOuts      263 non-null    float64
 14  Assists      263 non-null    float64
 15  Errors       263 non-null    float64
 16  League_N     263 non-null    uint8  
 17  Division_W   263 non-null    uint8  
 18  NewLeague_N  263 non-null    uint8  
dtypes: float64(16), uint8(3)
memory usage: 35.7 KB
```

1.2.4 Split data

Split the data set into train and test set with the first 70% of the data for training and the remaining 30% for testing.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
```

```
X_train.head()
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWa
260	496.0	119.0	8.0	57.0	33.0	21.0	7.0	3358.0	882.0	36.0	365.0	280.0	165.0
92	317.0	78.0	7.0	35.0	35.0	32.0	1.0	317.0	78.0	7.0	35.0	35.0	32.0
137	343.0	103.0	6.0	48.0	36.0	40.0	15.0	4338.0	1193.0	70.0	581.0	421.0	325.(
90	314.0	83.0	13.0	39.0	46.0	16.0	5.0	1457.0	405.0	28.0	156.0	159.0	76.0
100	495.0	151.0	17.0	61.0	84.0	78.0	10.0	5624.0	1679.0	275.0	884.0	1015.0	709.(

1.2.5 Standardization

Lasso performs best when all numerical features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

This means it is important to standardize our features. We do this by subtracting the mean from our observations and then dividing the difference by the standard deviation. This so called standard score z for an observation x is calculated as:

$$z = \frac{(x - \bar{x})}{s}$$

where:

- x is an observation in a feature
- $\bar{x}$ is the mean of that feature
- s is the standard deviation of that feature.

To avoid data leakage, the standardization of numerical features should always be performed after data splitting and only from training data. Furthermore, we obtain all necessary statistics for our features (mean and standard deviation) from training data and also use them on test data. Note that we don't standardize our dummy variables (which only have values of 0 or 1).

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X_train[list_numerical])

X_train[list_numerical] = scaler.transform(X_train[list_numerical])

X_test[list_numerical] = scaler.transform(X_test[list_numerical])
```

```
X_train
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits
260	0.644577	0.257439	-0.456963	0.101010	-0.763917	-0.975959	-0.070553	0.298535	0.239063
92	-0.592807	-0.671359	-0.572936	-0.778318	-0.685806	-0.458312	-1.306911	-1.001403	-0.969702
137	-0.413075	-0.105019	-0.688910	-0.258715	-0.646751	-0.081841	1.577925	0.717456	0.706633
90	-0.613545	-0.558091	0.122907	-0.618440	-0.256196	-1.211253	-0.482672	-0.514087	-0.478077
100	0.637665	0.982354	0.586803	0.260888	1.227914	1.706394	0.547626	1.267183	1.437305
...	...	...	...	...	...	...	...	...	...
274	0.824309	0.733164	0.470829	0.740521	0.954525	0.859335	-0.688732	-0.824858	-0.808834
196	0.423369	0.461321	1.862516	0.500704	1.618469	0.482865	1.165805	1.354814	1.246368
159	1.474109	1.254197	1.746542	1.140215	2.126191	-0.458312	-0.894792	-0.522636	-0.520174
17	-1.470728	-1.396275	-1.152806	-1.217982	-1.740306	-1.258312	-0.482672	-0.932153	-0.933620
162	-1.643547	-1.554850	-1.152806	-1.657646	-1.701250	-1.211253	-0.894792	-1.053127	-1.020819

184 rows × 19 columns

1.3 Lasso regression

First, we apply lasso regression on the training set with an arbitrarily regularization parameter α of 1.

```
from sklearn.linear_model import Lasso

reg = Lasso(alpha=1)
reg.fit(X_train, y_train)
```

```
Lasso(alpha=1)
```

1.3.1 Model evaluation

We print the R^2-score for the training and test set.

```
print('R squared training set', round(reg.score(X_train, y_train)*100, 2))
print('R squared test set', round(reg.score(X_test, y_test)*100, 2))
```

```
R squared training set 60.43
R squared test set 33.01
```

MSE for the training and test set.

```
from sklearn.metrics import mean_squared_error

# Training data
pred_train = reg.predict(X_train)
mse_train = mean_squared_error(y_train, pred_train)
print('MSE training set', round(mse_train, 2))

# Test data
pred = reg.predict(X_test)
mse_test =mean_squared_error(y_test, pred)
print('MSE test set', round(mse_test, 2))
```

```
MSE training set 80571.73
MSE test set 134426.33
```

1.4 Role of alpha

To better understand the role of alpha, we plot the lasso coefficients as a function of alpha (`max_iter` are the maximum number of iterations):

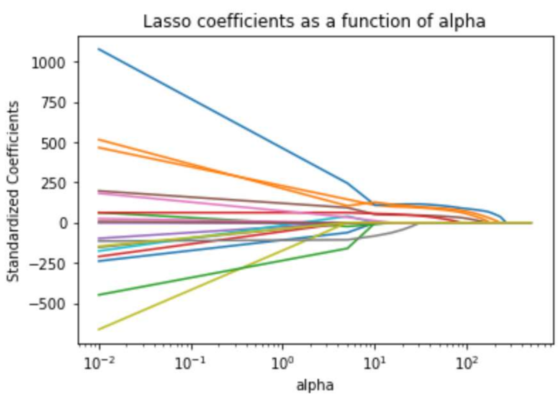
```
import numpy as np
import matplotlib.pyplot as plt

alphas = np.linspace(0.01,500,100)
lasso = Lasso(max_iter=10000)
coefs = []

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(X_train, y_train)
    coefs.append(lasso.coef_)

ax = plt.gca()

ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('Standardized Coefficients')
plt.title('Lasso coefficients as a function of alpha');
```



Remember that if alpha = 0, then the lasso gives the least squares fit, and when alpha becomes very large, the lasso gives the null model in which all coefficient estimates equal zero.

Moving from left to right in our plot, we observe that at first the lasso models contains many predictors with high magnitudes of coefficient estimates. With increasing alpha, the coefficient estimates approximate towards zero.

Next, we use cross-validation to find the best value for alpha.

1.5 Lasso with optimal alpha

To find the optimal value of alpha, we use scikit learns lasso linear model with iterative fitting along a regularization path (LassoCV). The best model is selected by cross-validation.

1.5.1 k-fold cross validation

```
from sklearn.linear_model import LassoCV

# Lasso with 5 fold cross-validation
model = LassoCV(cv=5, random_state=0, max_iter=10000)

# Fit model
model.fit(X_train, y_train)
```

```
LassoCV(cv=5, max_iter=10000, random_state=0)
```

Show best value of penalization chosen by cross validation:

```
model.alpha_
```

```
2.3441244939374593
```

1.5.2 Best model

Use best value for our final model:

```
# Set best alpha
lasso_best = Lasso(alpha=model.alpha_)
lasso_best.fit(X_train, y_train)
```

```
Lasso(alpha=2.3441244939374593)
```

Show model coefficients and names:

```
print(list(zip(lasso_best.coef_, X)))
```

```
[(-176.45309657050424, 'AtBat'), (271.23333276345227, 'Hits'), (-13.049492223041824, 'HmRun'
```

1.5.3 Model evaluation

```
print('R squared training set', round(lasso_best.score(X_train, y_train)*100, 2))
print('R squared test set', round(lasso_best.score(X_test, y_test)*100, 2))
```

```
R squared training set 59.18
R squared test set 35.48
```

```
mean_squared_error(y_test, lasso_best.predict(X_test))
```

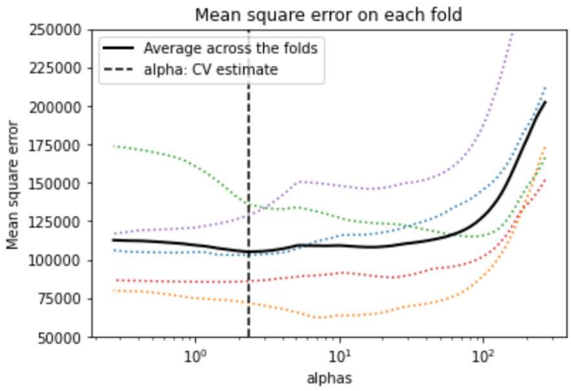
```
129468.59746481002
```

Lasso path: plot results of cross-validation with mean squared erros (for more information about the plot visit the scikit-learn documentation)

```python
plt.semilogx(model.alphas_, model.mse_path_, ":")
plt.plot(
    model.alphas_ ,
    model.mse_path_.mean(axis=-1),
    "k",
    label="Average across the folds",
    linewidth=2,
)
plt.axvline(
    model.alpha_, linestyle="--", color="k", label="alpha: CV estimate"
)

plt.legend()
plt.xlabel("alphas")
plt.ylabel("Mean square error")
plt.title("Mean square error on each fold")
plt.axis("tight")

ymin, ymax = 50000, 250000
plt.ylim(ymin, ymax);
```



Statistics

[Jan Kirenz](#)

Professor

I'm a data scientist educator and consultant.

Related

- [Lasso Regression with Python](#)
- [Linear regression diagnostics in Python](#)
- [Classification with Tidymodels, Workflows and Recipes](#)
- [Data Science with Tidymodels, Workflows and Recipes](#)
- [Applied Statistics](#)

LICENSE: CC-BY-SA

© Jan Kirenz, 2023 · Made with the blogdown package and the Academic theme for Hugo.