learn-co-curriculum / dsc-ridge-and-lasso-regression-lab Public

View license

0 stars 179 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 9 commits ahead, 10 commits behind master.

hoffm386 copy edits ... on Jun 30 13

View code

README.md

Ridge and Lasso Regression - Lab

Introduction

In this lab, you'll practice your knowledge of ridge and lasso regression!

Objectives

In this lab you will:

- Use lasso and ridge regression with scikit-learn
- Compare and contrast lasso, ridge and non-regularized regression

Housing Prices Data

We'll use this version of the Ames Housing dataset:

```python
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
df = pd.read_csv('housing_prices.csv', index_col=0)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   MSSubClass     1460 non-null   int64  
 1   MSZoning       1460 non-null   object 
 2   LotFrontage    1201 non-null   float64
 3   LotArea        1460 non-null   int64  
 4   Street         1460 non-null   object 
 5   Alley          91 non-null     object 
 6   LotShape       1460 non-null   object 
 7   LandContour    1460 non-null   object 
 8   Utilities      1460 non-null   object 
 9   LotConfig      1460 non-null   object 
 10  LandSlope      1460 non-null   object 
 11  Neighborhood   1460 non-null   object 
 12  Condition1     1460 non-null   object 
 13  Condition2     1460 non-null   object 
 14  BldgType       1460 non-null   object 
 15  HouseStyle     1460 non-null   object 
 16  OverallQual    1460 non-null   int64  
 17  OverallCond    1460 non-null   int64  
 18  YearBuilt      1460 non-null   int64  
 19  YearRemodAdd   1460 non-null   int64  
 20  RoofStyle      1460 non-null   object 
 21  RoofMatl       1460 non-null   object 
 22  Exterior1st    1460 non-null   object 
 23  Exterior2nd    1460 non-null   object 
 24  MasVnrType     1452 non-null   object 
 25  MasVnrArea     1452 non-null   float64
 26  ExterQual      1460 non-null   object 
 27  ExterCond      1460 non-null   object 
 28  Foundation     1460 non-null   object 
 29  BsmtQual       1423 non-null   object 
 30  BsmtCond       1423 non-null   object 
 31  BsmtExposure   1422 non-null   object 
 32  BsmtFinType1   1423 non-null   object 
 33  BsmtFinSF1     1460 non-null   int64  
 34  BsmtFinType2   1422 non-null   object 
 35  BsmtFinSF2     1460 non-null   int64  
```

```
 36  BsmtUnfSF      1460 non-null   int64
 37  TotalBsmtSF    1460 non-null   int64
 38  Heating        1460 non-null   object
 39  HeatingQC      1460 non-null   object
 40  CentralAir     1460 non-null   object
 41  Electrical     1459 non-null   object
 42  1stFlrSF       1460 non-null   int64
 43  2ndFlrSF       1460 non-null   int64
 44  LowQualFinSF   1460 non-null   int64
 45  GrLivArea      1460 non-null   int64
 46  BsmtFullBath   1460 non-null   int64
 47  BsmtHalfBath   1460 non-null   int64
 48  FullBath       1460 non-null   int64
 49  HalfBath       1460 non-null   int64
 50  BedroomAbvGr   1460 non-null   int64
 51  KitchenAbvGr   1460 non-null   int64
 52  KitchenQual    1460 non-null   object
 53  TotRmsAbvGrd   1460 non-null   int64
 54  Functional     1460 non-null   object
 55  Fireplaces     1460 non-null   int64
 56  FireplaceQu    770 non-null    object
 57  GarageType     1379 non-null   object
 58  GarageYrBlt    1379 non-null   float64
 59  GarageFinish   1379 non-null   object
 60  GarageCars     1460 non-null   int64
 61  GarageArea     1460 non-null   int64
 62  GarageQual     1379 non-null   object
 63  GarageCond     1379 non-null   object
 64  PavedDrive     1460 non-null   object
 65  WoodDeckSF     1460 non-null   int64
 66  OpenPorchSF    1460 non-null   int64
 67  EnclosedPorch  1460 non-null   int64
 68  3SsnPorch      1460 non-null   int64
 69  ScreenPorch    1460 non-null   int64
 70  PoolArea       1460 non-null   int64
 71  PoolQC         7 non-null      object
 72  Fence          281 non-null    object
 73  MiscFeature    54 non-null     object
 74  MiscVal        1460 non-null   int64
 75  MoSold         1460 non-null   int64
 76  YrSold         1460 non-null   int64
 77  SaleType       1460 non-null   object
 78  SaleCondition  1460 non-null   object
 79  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(34), object(43)
memory usage: 923.9+ KB
```

More information about the features is available in the `data_description.txt` file in this repository.

Data Preparation

The code below:

- Separates the data into `X` (predictor) and `y` (target) variables
- Splits the data into 75-25 training-test sets, with a `random_state` of 10
- Separates each of the `X` values into continuous vs. categorical features
- Fills in missing values (using different strategies for continuous vs. categorical features)
- Scales continuous features to a range of 0 to 1
- Dummy encodes categorical features
- Combines the preprocessed continuous and categorical features back together

```
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

# Create X and y
y = df['SalePrice']
X = df.drop(columns=['SalePrice'])

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)

# Separate X data into continuous vs. categorical
X_train_cont = X_train.select_dtypes(include='number')
X_test_cont = X_test.select_dtypes(include='number')
X_train_cat = X_train.select_dtypes(exclude='number')
X_test_cat = X_test.select_dtypes(exclude='number')

# Impute missing values using SimpleImputer, median for continuous and
# filling in 'missing' for categorical
impute_cont = SimpleImputer(strategy='median')
X_train_cont = impute_cont.fit_transform(X_train_cont)
X_test_cont = impute_cont.transform(X_test_cont)
impute_cat = SimpleImputer(strategy='constant', fill_value='missing')
X_train_cat = impute_cat.fit_transform(X_train_cat)
X_test_cat = impute_cat.transform(X_test_cat)

# Scale continuous values using MinMaxScaler
```

```
scaler = MinMaxScaler()
X_train_cont = scaler.fit_transform(X_train_cont)
X_test_cont = scaler.transform(X_test_cont)

# Dummy encode categorical values using OneHotEncoder
ohe = OneHotEncoder(handle_unknown='ignore')
X_train_cat = ohe.fit_transform(X_train_cat)
X_test_cat = ohe.transform(X_test_cat)

# Combine everything back together
X_train_preprocessed = np.concatenate([X_train_cont, X_train_cat.todense()], axis=1)
X_test_preprocessed = np.concatenate([X_test_cont, X_test_cat.todense()], axis=1)
```

Linear Regression Model

Let's use this data to build a first naive linear regression model. Fit the model on the training data (`X_train_preprocessed`), then compute the R-Squared and the MSE for both the training and test sets.

```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression

# Fit the model
linreg = LinearRegression()
linreg.fit(X_train_preprocessed, y_train)

# Print R2 and MSE for training and test sets
print('Training r^2:', linreg.score(X_train_preprocessed, y_train))
print('Test r^2:    ', linreg.score(X_test_preprocessed, y_test))
print('Training MSE:', mean_squared_error(y_train, linreg.predict(X_train_preprocess
print('Test MSE:    ', mean_squared_error(y_test, linreg.predict(X_test_preprocessed
```

```
Training r^2: 0.935894474732577
Test r^2:     -2.51677622190316e+19
Training MSE: 402648057.96712327
Test MSE:     1.6057498174589344e+29
```

Notice the severe overfitting above; our training R-Squared is very high, but the test R-Squared is negative! Similarly, the scale of the test MSE is orders of magnitude higher than that of the training MSE.

Ridge and Lasso Regression

Use all the data (scaled features and dummy categorical variables, `X_train_preprocessed`) to build some models with regularization - two each for lasso and ridge regression. Each time, look at R-Squared and MSE.

Remember that you can use the scikit-learn documentation if you don't remember how to import or use these classes:

- `Lasso` documentation
- `Ridge` documentation

Lasso

With default hyperparameters (`alpha` = 1)

```
from sklearn.linear_model import Lasso

lasso = Lasso() # Lasso is also known as the L1 norm
lasso.fit(X_train_preprocessed, y_train)

print('Training r^2:', lasso.score(X_train_preprocessed, y_train))
print('Test r^2:    ', lasso.score(X_test_preprocessed, y_test))
print('Training MSE:', mean_squared_error(y_train, lasso.predict(X_train_preprocesse
print('Test MSE:    ', mean_squared_error(y_test, lasso.predict(X_test_preprocessed)
```

```
Training r^2: 0.9358329893282811
Test r^2:     0.8894619546974663
Training MSE: 403034248.99402535
Test MSE:     705253190.6574916
```

With a higher regularization hyperparameter (`alpha` = 10)

```
lasso_10 = Lasso(alpha=10)
lasso_10.fit(X_train_preprocessed, y_train)

print('Training r^2:', lasso_10.score(X_train_preprocessed, y_train))
print('Test r^2:    ', lasso_10.score(X_test_preprocessed, y_test))
```

```
print('Training MSE:', mean_squared_error(y_train, lasso_10.predict(X_train_preproce
print('Test MSE:     ', mean_squared_error(y_test, lasso_10.predict(X_test_preprocess
```

```
Training r^2: 0.9340791643883491
Test r^2:     0.8980800909835147
Training MSE: 414050057.7426786
Test MSE:     650267885.8547701
```

Ridge

With default hyperparameters (alpha = 1)

```
from sklearn.linear_model import Ridge

ridge = Ridge() # Ridge is also known as the L2 norm
ridge.fit(X_train_preprocessed, y_train)

print('Training r^2:', ridge.score(X_train_preprocessed, y_train))
print('Test r^2:     ', ridge.score(X_test_preprocessed, y_test))
print('Training MSE:', mean_squared_error(y_train, ridge.predict(X_train_preprocesse
print('Test MSE:     ', mean_squared_error(y_test, ridge.predict(X_test_preprocessed)
```

```
Training r^2: 0.920803998083458
Test r^2:     0.8863596364585585
Training MSE: 497431636.93662256
Test MSE:     725046555.2898804
```

With higher regularization hyperparameter (alpha = 10)

```
ridge_10 = Ridge(alpha=10)
ridge_10.fit(X_train_preprocessed, y_train)

print('Training r^2:', ridge_10.score(X_train_preprocessed, y_train))
print('Test r^2:     ', ridge_10.score(X_test_preprocessed, y_test))
print('Training MSE:', mean_squared_error(y_train, ridge_10.predict(X_train_preproce
print('Test MSE:     ', mean_squared_error(y_test, ridge_10.predict(X_test_preprocess
```

```
Training r^2: 0.8889512334535105
Test r^2:     0.8794931799899866
Training MSE: 697499474.547024
Test MSE:     768855818.604763
```

Comparing the Metrics

Which model seems best, based on the metrics?

```python
print('Test r^2')
print('Linear Regression:', linreg.score(X_test_preprocessed, y_test))
print('Lasso, alpha=1:   ', lasso.score(X_test_preprocessed, y_test))
print('Lasso, alpha=10:  ', lasso_10.score(X_test_preprocessed, y_test))
print('Ridge, alpha=1:   ', ridge.score(X_test_preprocessed, y_test))
print('Ridge, alpha=10:  ', ridge_10.score(X_test_preprocessed, y_test))
print()
print('Test MSE')
print('Linear Regression:', mean_squared_error(y_test, linreg.predict(X_test_preproc
print('Lasso, alpha=1:   ', mean_squared_error(y_test, lasso.predict(X_test_preproce
print('Lasso, alpha=10:  ', mean_squared_error(y_test, lasso_10.predict(X_test_prepr
print('Ridge, alpha=1:   ', mean_squared_error(y_test, ridge.predict(X_test_preproce
print('Ridge, alpha=10:  ', mean_squared_error(y_test, ridge_10.predict(X_test_prepr
```

```
Test r^2
Linear Regression: -2.51677622190316e+19
Lasso, alpha=1:    0.8894619546974663
Lasso, alpha=10:   0.8980800909835147
Ridge, alpha=1:    0.8863596364585585
Ridge, alpha=10:   0.8794931799899866

Test MSE
Linear Regression: 1.6057498174589344e+29
Lasso, alpha=1:    705253190.6574916
Lasso, alpha=10:   650267885.8547701
Ridge, alpha=1:    725046555.2898804
Ridge, alpha=10:   768855818.604763
```

▶ **Answer (click to reveal)**

Comparing the Parameters

Compare the number of parameter estimates that are (very close to) 0 for the `Ridge` and `Lasso` models with `alpha` =10.

Use 10**(-10) as an estimate that is very close to 0.

```
print('Zeroed-out ridge params:', sum(abs(ridge_10.coef_) < 10**(-10)),
     'out of', len(ridge_10.coef_))
```

```
Zeroed-out ridge params: 0 out of 295
```

```
print('Zeroed-out lasso params:', sum(abs(lasso_10.coef_) < 10**(-10)),
     'out of', len(lasso_10.coef_))
```

```
Zeroed-out lasso params: 82 out of 295
```

▶ **Answer (click to reveal)**

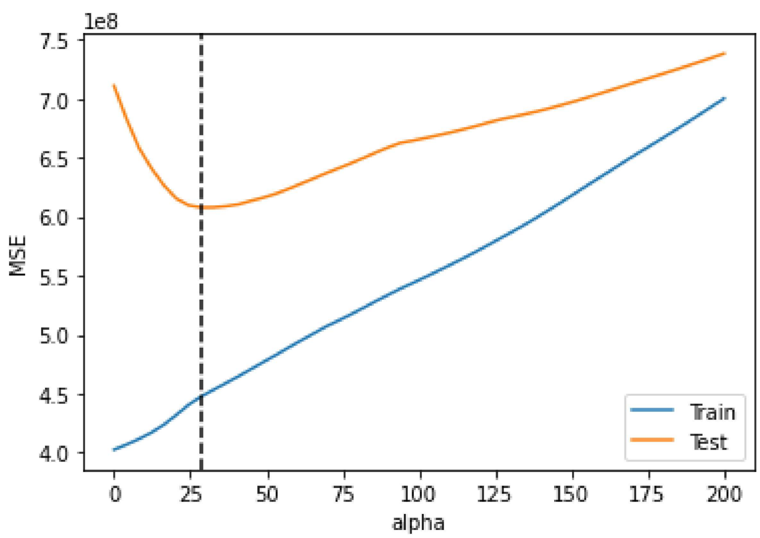
Finding an Optimal Alpha

Earlier we tested two values of `alpha` to see how it affected our MSE and the value of our coefficients. We could continue to guess values of `alpha` for our ridge or lasso regression one at a time to see which values minimize our loss, or we can test a range of values and pick the alpha which minimizes our MSE. Here is an example of how we would do this:

```
import matplotlib.pyplot as plt
%matplotlib inline

train_mse = []
test_mse = []
alphas = np.linspace(0, 200, num=50)

for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train_preprocessed, y_train)

    train_preds = lasso.predict(X_train_preprocessed)
    train_mse.append(mean_squared_error(y_train, train_preds))
```

```
    test_preds = lasso.predict(X_test_preprocessed)
    test_mse.append(mean_squared_error(y_test, test_preds))

fig, ax = plt.subplots()
ax.plot(alphas, train_mse, label='Train')
ax.plot(alphas, test_mse, label='Test')
ax.set_xlabel('alpha')
ax.set_ylabel('MSE')

# np.argmin() returns the index of the minimum value in a list
optimal_alpha = alphas[np.argmin(test_mse)]

# Add a vertical line where the test MSE is minimized
ax.axvline(optimal_alpha, color='black', linestyle='--')
ax.legend();

print(f'Optimal Alpha Value: {int(optimal_alpha)}')
```

```
Optimal Alpha Value: 28
```



Take a look at this graph of our training and test MSE against `alpha`. Try to explain to yourself why the shapes of the training and test curves are this way. Make sure to think about what `alpha` represents and how it relates to overfitting vs underfitting.

▶ **Answer (click to reveal)**

Summary

Well done! You now know how to build lasso and ridge regression models, use them for feature selection and find an optimal value for `alpha` .

Releases

No releases published

Packages

No packages published

Contributors 7

Languages

- Jupyter Notebook 100.0%