AVADHUT VARVATKAR · 9MO AGO · 138 VIEWS

Ridge Regression

Python · No attached data sources

Ridge Regression

Ridge regression, also known as Tikhonov regularization, is a technique used in statistical regression analysis to deal with the problem of multicollinearity, where the independent variables are highly correlated with each other. It is an extension of ordinary least squares (OLS) regression.

In ridge regression, a penalty term is added to the OLS objective function to shrink the coefficient estimates towards zero. This penalty term is proportional to the square of the magnitudes of the coefficients, multiplied by a tuning parameter lambda (λ). The larger the value of λ, the greater the amount of shrinkage applied to the coefficients.

The objective function of ridge regression can be represented as:

minimize ||Y - Xβ||^2 + λ||β||^2,

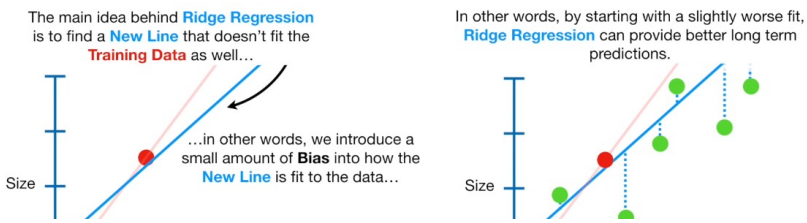
where:

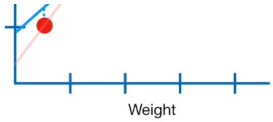
- Y is the dependent variable,
- X is the matrix of independent variables,
- β is the vector of coefficients to be estimated,
- λ is the tuning parameter that controls the amount of shrinkage.

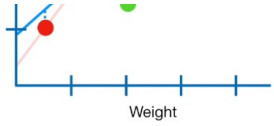
Ridge regression has several advantages:

1. It reduces the impact of multicollinearity by shrinking the coefficients towards zero, leading to more stable and reliable estimates.
2. It can handle situations where the number of predictors (independent variables) is larger than the number of observations.
3. Ridge regression can still provide useful results even when the predictors are highly correlated.

techniques, such as k-fold cross-validation, can be used to select the optimal value of λ that balances model complexity and prediction accuracy.







Ridge in Simple linear Regression

It's used to reduce the slope of line or coefficient of line. In other words, we increase a small amount of Bias into how the new line is fit to the data.

```
Formula of simple linear regression
```

$$\boldsymbol{y = mx + c}$$

Try to reduce m_slop

```
Lose function in simple ridge Regression:
```

minimize

- $$\boldsymbol{L = \sum(y_i - \widehat{y_i})^2 \ + \ \lambda m^2}$$

- y is the Training variable,
- y^is the Predicted variables,
- m is the vector of coefficients to be estimated,
- λ is the tuning parameter that controls the amount of shrinkage.

Now derivate the lose function with respect to m and b
The Derivate with respected to b is:

$$\boldsymbol{: \ b = \ \bar{y} - m\bar{x}}$$

where

- y¯is the mean of all y variable,
- x¯is the mean of all x variables,
- m is the vector of coefficients to be estimated,

Putting value of b in lose function:

$$\boldsymbol{L} = \sum(y_i - mx_i - \ \bar{y} - m\bar{x})^2 \ + \ \lambda\, m^2$$

$$\frac{\partial l}{\partial m} = \frac{\partial}{\partial m}\,(\sum(y_i - mx_i - \ \bar{y} - m\bar{x})^2 \ + \ \lambda\, m^2$$

$$= \ -2\sum(y_i - mx_i - m\bar{x})(x_i - \bar{x}\,) \quad + \ 2\,\lambda\, m$$

$$= \ \lambda\, m - \sum(y_i - \ \bar{y})(x_i - \bar{x}) - \ m(x_i - \bar{x})\text{^}2$$

$$\sum(y_i - \bar{y})(x_i - \bar{x})$$

- $$m = \frac{\sum(y_i - \bar{y})(x_i - \bar{x})}{\sum\left((x_i - \bar{x})\right)^2 + \lambda}$$

Where:_

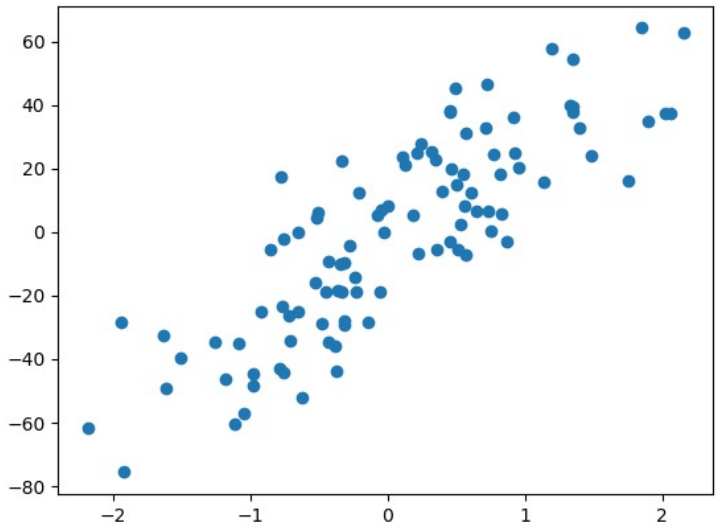
- y is the Training dependent variable,
- x is the Training independent variable
- $\bar{y}$ is the mean of all y variable,
- $\bar{x}$ is the mean of all x variables,
- m is the vector of coefficients to be estimated,
- λ is the tuning parameter that controls the amount of shrinkage.

In [1]:
```
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
X,y = make_regression(n_samples=100, n_features=1, n_informative=1, n_t
argets=1,noise=20,random_state=13)
```

In [3]:
```
plt.scatter(X,y)
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x7a8049908e80>
```



In [4]:
```
from sklearn.linear_model import LinearRegression
```

In [5]:
```
lr = LinearRegression()
lr.fit(X,y)
print(lr.coef_)
print(lr.intercept_)
```

```
[27.82809103]
-2.29474455867698
```

In [6]:

```
from sklearn.linear_model import Ridge
```

In [7]:

```
rr = Ridge(alpha=10)
rr.fit(X,y)
print(rr.coef_)
print(rr.intercept_)
```

```
[24.9546267]
-2.1269130035235735
```

In [8]:

```
rr1 = Ridge(alpha=50)
rr1.fit(X,y)
print(rr1.coef_)
print(rr1.intercept_)
```

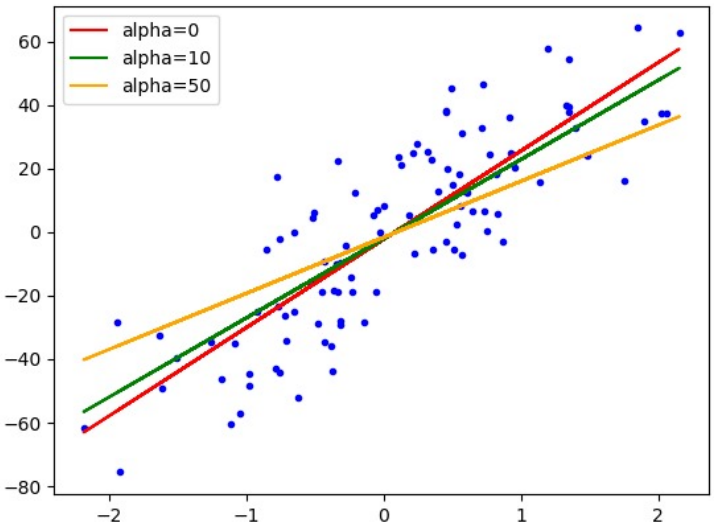
```
[17.66035643]
-1.7008737066555062
```

In [9]:

```
plt.plot(X,y,'b.')
plt.plot(X,lr.predict(X),color='red',label='alpha=0')
plt.plot(X,rr.predict(X),color='green',label='alpha=10')
plt.plot(X,rr1.predict(X),color='orange',label='alpha=50')
plt.legend()
```

Out[9]:

```
<matplotlib.legend.Legend at 0x7a8046ef89a0>
```



In [10]:

```
def linear_regression(X,y,alpha=1):
    x_mean = X.mean()
    y_mean = y.mean()

    num = 0
    den = 0

    for i in range(X.shape[0]):
        num = num + (y[i] - y_mean) * (X[i] - x_mean)
        den = den + (X[i] - x_mean) * (X[i] - x_mean)

    m = num/(den + alpha)
    b = y_mean - m*x_mean

    return m,b
```

In [11]:

```
class MeraRidge:

    def __init__(self,alpha=0.1):
        self.alpha = alpha
        self.m = None
        self.b = None

    def fit(self,X_train,y_train):

        num = 0
        den = 0

        for i in range(X_train.shape[0]):
            num = num + (y_train[i] - y_train.mean())*(X_train[i] - X_t
rain.mean())
            den = den + (X_train[i] - X_train.mean())*(X_train[i] - X_t
rain.mean())

        self.m = num/(den + self.alpha)
        self.b = y_train.mean() - (self.m*X_train.mean())
        print(self.m,self.b)

    def predict(X_test):
        return self.m*x + self.b
```

In [12]:

```
reg = MeraRidge(alpha=100)
```

In [13]:

```
reg.fit(X,y)
```

```
[12.93442104] [-1.42484415]
```

Multiple linear Regression

Multiple ridge regression, also known as multivariate ridge regression, is an extension of ridge regression that allows for the analysis of multiple dependent variables simultaneously.

It is used when there are multiple response variables that are correlated with each other and with the independent variables.

In multiple ridge regression, the objective is to estimate the regression coefficients that minimize the sum of squared errors for all the response variables, while also incorporating the ridge penalty term. The objective function can be represented as:

In Multiple regression is lose function is :

$$L = (xw - y)^T (xw - y)$$

)

where:

- Y is the dependent variable,
- X is the matrix of independent variables,
- w is the vector of coefficients to be estimated,
- λ is the tuning parameter that controls the amount of shrinkage.

Lose function for Ridge Regression

$$L = (xw - y)^T (xw - y) + \lambda w^T w$$
$$L = \left(x^T w^T - y^T\right) \quad (xw - y) + \lambda w^T w$$

Arrows Multiplication

$$= w^T x^T xw - 2w^T x^T y + y^t y + \lambda \mathrm{w}^{\mathrm{t}} w$$

Derivate the equation

$$w = \left(\boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x} + \boldsymbol{\lambda}\, \mathbf{I}\right)^{-\mathbf{1}} * \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{y}$$

```
where:
```

- Y is the dependent variable,
- X is the matrix of independent variables,
- w is the vector of coefficients to be estimated,
- I is the Identity Metrics
- λ is the tuning parameter that controls the amount of shrinkage.

In [14]:

```
from sklearn.datasets import load_diabetes

data=load_diabetes()
```

In [15]:

```
print(data.DESCR)
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of
n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive val
ues

  :Target: Column 11 is a quantitative measure of disease progressio
n one year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and
scaled by the standard deviation times the square root of `n_samples
` (i.e. the sum of squares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani
(2004) "Least Angle Regression," Annals of Statistics (with discussi
on), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

In [16]:

```
X=data.data
y=data.target
```

In [17]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random
_state=45)
```

In [18]:

```
from sklearn.linear_model import LinearRegression
L=LinearRegression()
```

In [19]:
```
L.fit(X_train,y_train)
```

Out[19]:
```
▾ LinearRegression
LinearRegression()
```

In [20]:
```
print(L.coef_)
print(L.intercept_)
```

```
[  23.45465406 -247.42747406  492.1087518   329.35876431 -970.797230
39
  573.54295519  182.42162368  255.92168168  794.21609282   89.322492
14]
152.13623331746496
```

In [21]:
```
y_pred=L.predict(X_test)
```

In [22]:
```
from sklearn.metrics import r2_score,mean_squared_error

print("R2 score",r2_score(y_test,y_pred))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
R2 score 0.5188113124539249
RMSE 48.72713760953253
```

In [23]:
```
from sklearn.linear_model import Ridge
R=Ridge(alpha=100000)
```

In [24]:
```
R.fit(X_train,y_train)
```

Out[24]:
```
▾ Ridge
Ridge(alpha=100000)
```

In [25]:
```
print(R.coef_)
print(R.intercept_)
```

```
[ 0.00260126  0.00057066  0.00776597  0.0060976   0.00233864  0.0018
4724
 -0.00513942  0.0052716   0.00734598  0.00528629]
151.83287930791352
```

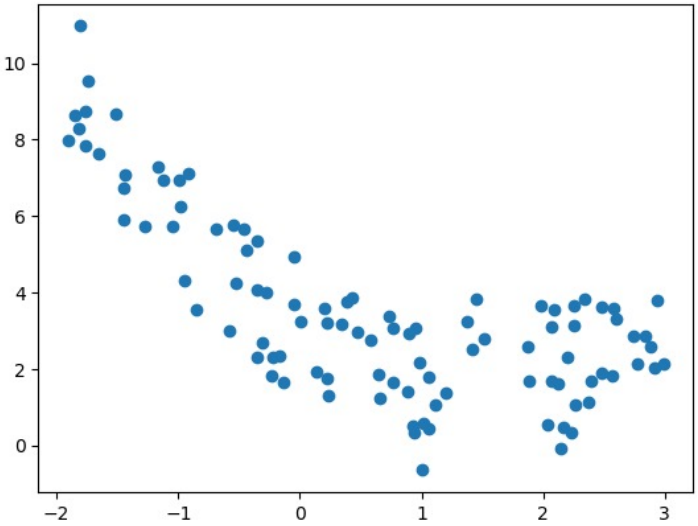
In [26]:
```
y_pred1=R.predict(X_test)
```

In [27]:
```
print("R2 score",r2_score(y_test,y_pred1))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred1)))
```

```
R2 score -0.00042490200441935855
RMSE 70.25956272917782
```

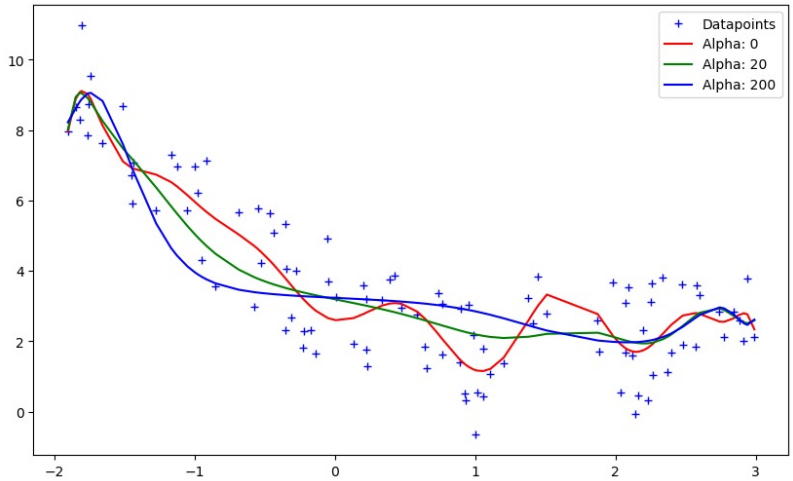
In [28]:

```
m = 100
x1 = 5 * np.random.rand(m, 1) - 2
x2 = 0.7 * x1 ** 2 - 2 * x1 + 3 + np.random.randn(m, 1)

plt.scatter(x1, x2)
plt.show()
```



In [29]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

def get_preds_ridge(x1, x2, alpha):
    model = Pipeline([
        ('poly_feats', PolynomialFeatures(degree=16)),
        ('ridge', Ridge(alpha=alpha))
    ])
    model.fit(x1, x2)
    return model.predict(x1)

alphas = [0, 20, 200]
cs = ['r', 'g', 'b']

plt.figure(figsize=(10, 6))
plt.plot(x1, x2, 'b+', label='Datapoints')

for alpha, c in zip(alphas, cs):
    preds = get_preds_ridge(x1, x2, alpha)
    # Plot
    plt.plot(sorted(x1[:, 0]), preds[np.argsort(x1[:, 0])], c, label='A
lpha: {}'.format(alpha))

plt.legend()
plt.show()
```



In [30]:

```
class MyRidge:

    def __init__(self,alpha=0.1):

        self.alpha = alpha
        self.coef_ = None
        self.intercept_ = None

    def fit(self,X_train,y_train):

        X_train = np.insert(X_train,0,1,axis=1)
        I = np.identity(X_train.shape[1])
        I[0][0] = 0
        result = np.linalg.inv(np.dot(X_train.T,X_train) + self.alpha *
I).dot(X_train.T).dot(y_train)
        self.intercept_ = result[0]
        self.coef_ = result[1:]

    def predict(self,X_test):
        return np.dot(X_test,self.coef_) + self.intercept_
```

In [31]:

```
reg = MyRidge()
reg.fit(X_train,y_train)
y_pred = reg.predict(X_test)
print(r2_score(y_test,y_pred))
print(reg.coef_)
print(reg.intercept_)
```

```
0.5208421054129914
[  31.03389163 -204.8305909   464.8963454   304.96414918  -95.760370
39
  -87.52969228 -183.81809293  147.49794012  425.85392451  110.215599
15]
152.0714129017282
```

Gradient Descent Ridge

Gradient descent ridge regression, also known as ridge regression with gradient descent, is a variant of ridge regression that utilizes the gradient descent optimization algorithm to estimate the regression coefficients. It combines the concept of ridge regression with the iterative nature of gradient descent to find the optimal values of the coefficients. In gradient descent ridge regression, the objective is still to minimize the sum of squared errors, but

with the additional ridge penalty term. The gradient descent algorithm is employed to iteratively update the coefficient estimates by taking steps in the direction of steepest descent of the objective function.

The steps involved in gradient descent ridge regression are as follows:

1. Initialize the coefficient values (β) to some arbitrary values.
2. Calculate the gradient of the objective function with respect to the coefficients.
3. Update the coefficient estimates by taking a step in the direction of the negative gradient, multiplied by a learning rate (α).
4. Repeat steps 2 and 3 until convergence or a predetermined number of iterations.

The ridge penalty term is typically incorporated into the gradient descent updates by adding the ridge penalty term to the gradient calculation. This penalty term helps to shrink the coefficient estimates towards zero and reduce the impact of multicollinearity. The learning rate (α) in gradient descent controls the size of the steps taken during each iteration. It is an important hyperparameter that needs to be carefully chosen. If the learning rate is too large, the algorithm may fail to converge, while if it is too small, the convergence may be slow.

Gradient descent ridge regression can be computationally efficient, especially when dealing with large datasets or a large number of predictors. However, it requires careful tuning of hyper parameters, such as the learning rate and the regularization parameter λ, to ensure convergence and find the optimal solution. It's important to note that there are other optimization algorithms available for ridge regression, such as coordinate descent and singular value decomposition (SVD), which may offer advantages in different scenarios.

$$\boldsymbol{W_{new} = W_{old} - \eta \frac{\Delta \mathbf{L}}{\Delta w}}$$

$$L = \frac{1}{2}\left(x^T w^T - y^T\right) \quad (xw - y) + \frac{1}{2}\lambda\, w^T w$$

Derivate the Equation:

$$\boldsymbol{\frac{\Delta L}{\Delta w} = \quad x^T x w \; - x^T y + \; \lambda\, \mathbf{w}}$$

In [32]:
```
from sklearn.linear_model import SGDRegressor
```

In [33]:
```
reg = SGDRegressor(penalty='l2',max_iter=500,eta0=0.1,learning_rate='co
nstant',alpha=0.001)
```

In [34]:
```
reg.fit(X_train,y_train)

y_pred = reg.predict(X_test)
print("R2 score",r2_score(y_test,y_pred))
print(reg.coef_)
print(reg.intercept_)
```

```
R2 score 0.4785293125926354
[  38.22515096 -133.82744301  383.70161572  255.35449204  -29.423522
49
  -75.36578773 -179.96913145  131.16255901  329.04852807  130.065463
69]
[163.03551361]
```

In [35]:
```
reg = Ridge(alpha=0.001, max_iter=500,solver='sparse_cg')
```

In [36]:

```
reg.fit(X_train,y_train)

y_pred = reg.predict(X_test)
print("R2 score",r2_score(y_test,y_pred))
print(reg.coef_)
print(reg.intercept_)
```

```
R2 score 0.5201448363733796
[  24.01614157 -246.40604595  493.59285633  329.08832668 -852.767740
04
  479.42466511  131.86683001  243.03291514  748.9646394    90.155078
92]
152.12463295186845
```

In [37]:

```
class RidgeGD:

    def __init__(self,epochs,learning_rate,alpha):

        self.learning_rate = learning_rate
        self.epochs = epochs
        self.alpha = alpha
        self.coef_ = None
        self.intercept_ = None

    def fit(self,X_train,y_train):

        self.coef_ = np.ones(X_train.shape[1])
        self.intercept_ = 0
        thetha = np.insert(self.coef_,0,self.intercept_)

        X_train = np.insert(X_train,0,1,axis=1)

        for i in range(self.epochs):
            thetha_der = np.dot(X_train.T,X_train).dot(thetha) - np.dot
(X_train.T,y_train) + self.alpha*thetha
            thetha = thetha - self.learning_rate*thetha_der

        self.coef_ = thetha[1:]
        self.intercept_ = thetha[0]

    def predict(self,X_test):

        return np.dot(X_test,self.coef_) + self.intercept_
```

In [38]:

```
reg = RidgeGD(epochs=500,alpha=0.001,learning_rate=0.005)
```

In [39]:

```
reg.fit(X_train,y_train)

y_pred = reg.predict(X_test)
print("R2 score",r2_score(y_test,y_pred))
print(reg.coef_)
print(reg.intercept_)
```

```
R2 score 0.515430712996193
[  34.15325939 -186.67106566  467.12291202  305.9828481   -54.219864
74
 -113.96322748 -209.37110173  140.75794314  397.61388652  140.494620
69]
152.0822237478433
```

License

Continue exploring

Input
1 file

Output
0 files

Logs
31.3 second run - successful

Comments
0 comments