MahdiAnvari_Homework01

March 8, 2024

[1]:
```
# Mahdi Anvari 610700002
import numpy as np
from sklearn.linear_model import LinearRegression, Lasso
```

[2]:
```
# question 1.a
# It generates synthetic data for a linear regression problem :
# a design matrix X, a coefficient vector Beta, and adds noise to simulate a␣
↪response vector Y.

def GenerateData(n, p,q):
    X = np.random.normal(size=(n, p))
    Beta = np.random.normal(size=(p, 1))
    SelectedColumns = np.random.choice(p, q, replace=False)
    SelectedX = X[:, SelectedColumns]
    SelectedBeta = Beta[SelectedColumns]
    NewBeta = np.zeros_like(Beta)
    NewBeta[SelectedColumns] = SelectedBeta
    Epsilon = np.random.normal(scale=0.05, size=(n, 1))
    Y = X.dot(NewBeta) + Epsilon
    return(Y,X,NewBeta)

Data = GenerateData(10,5,3)
y = Data[0]
x = Data[1]
Beta = Data[2]
print(Beta)
```

```
[[ 1.79596531]
 [-0.23970299]
 [ 0.        ]
 [ 0.        ]
 [ 0.37734638]]
```

[3]:
```
# question 1.b
# It computes the coefficients of a linear regression model using the␣
↪closed-form solution

def ClosedForm(X, Y):
```

```
    Beta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
    return(Beta)

print(ClosedForm(x,y))
```

```
[[ 1.80277949]
 [-0.24605351]
 [-0.00296861]
 [ 0.02103424]
 [ 0.36499013]]
```

[4]:

```
# question 1.c
# It performs gradient descent optimization to estimate the coefficients of a␣
 ↪linear regression model

def GradientDescent(X,Y,Alpha=0.001,TH=0.0001,MaxIter=10000):
    n = np.shape(X)[1]
    p = np.shape(X)[1]
    Beta = np.zeros((p, 1))
    PrevCost = float('inf')
    for _ in range(MaxIter):
        PredY = X.dot(Beta)
        Error = PredY - Y
        Cost = np.sum(Error ** 2)/2
        Gradient = X.T.dot(Error)
        Beta -= Alpha * Gradient
        if abs(Cost-PrevCost) < TH:
            break
        PrevCost = Cost
    return(Beta)

print(GradientDescent(x,y))
```

```
[[ 1.73727378]
 [-0.21001671]
 [-0.02304932]
 [ 0.03147785]
 [ 0.30811234]]
```

[5]:

```
# question 1.d
# It utilizes scikit-learn's LinearRegression model to estimate the␣
 ↪coefficients of a linear regression model

def SciKitLinearReg(X,Y):
    n = np.shape(X)[1]
    Model = LinearRegression()
    Model.fit(X,Y)
    return np.array(Model.coef_).reshape(n,1)
```

```
print(SciKitLinearReg(x,y))
```

```
[[ 1.81845799e+00]
 [-2.46773301e-01]
 [-1.01382127e-03]
 [ 2.55283818e-02]
 [ 3.69818868e-01]]
```

[6]:

```
# question 1.e
# It implements the LASSO regression algorithm
# Within the function, both X and Y are normalized
# In each iteration, it updates the coefficient estimates Beta by considering␣
↪the L1 penalty term and the gradient of the cost function

def NormalizeVector(vector):
    mean = np.mean(vector)
    normalized_vector = vector - mean
    norm = np.linalg.norm(normalized_vector)
    return normalized_vector / norm

def NormalizeMatrix(matrix):
    col_means = np.mean(matrix, axis=0)
    normalized_matrix = matrix - col_means
    col_norms = np.linalg.norm(normalized_matrix, axis=0)
    return normalized_matrix / col_norms

def LASSOreg(X,Y,Lambda=0.0001,TH=0.0001,MaxIter=1000):
    X = NormalizeMatrix(X)
    Y = NormalizeMatrix(Y)
    p = np.shape(X)[1]
    Beta = np.zeros((p, 1))
    PrevCost = float('inf')
    for _ in range(MaxIter):
        for i in range(p):
            PredY = X.dot(Beta)
            XK = (X[:, i])
            Error = Y - PredY + (XK*Beta[i]).reshape(10,1)
            NewXK = (XK.T.reshape(10,1))
            Gradient = NewXK.T.dot(Error)
            if Gradient < -Lambda:
                Beta[i] = (Gradient + Lambda)
            elif Gradient > Lambda:
                Beta[i] = (Gradient - Lambda)
            else:
                Beta[i] = 0
        Cost = np.sum(Error ** 2) / 2
```

```
        if abs(Cost-PrevCost) < TH:
            break
        PrevCost = Cost
    return(Beta)

print(LASSOreg(x,y))
```

```
[[ 1.21446822e+00]
 [-2.28894495e-01]
 [-8.59626072e-04]
 [ 2.47606844e-02]
 [ 3.58627606e-01]]
```

[7]:
```
# question 1.f
# It utilizes scikit-learn's Lasso model to perform LASSO regression

def SciKitLASSO(X,Y):
    n = np.shape(X)[1]
    Model = Lasso(alpha=0.001 , max_iter=1000)
    Model.fit(X,Y)
    return np.array(Model.coef_).reshape(n,1)

print(SciKitLASSO(x,y))
```

```
[[ 1.81229170e+00]
 [-2.44645290e-01]
 [-4.31411920e-04]
 [ 2.43587663e-02]
 [ 3.66703533e-01]]
```

[8]:
```
# question 2
```

1 question 2.a

Mahdi Anvari

Year:........ Month:........ Day:........

2.a) $\min_\beta \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \rightarrow J(\beta) = (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta$

$$y - X\beta = \begin{bmatrix} y_1 \\ \cdots \\ \cdots \\ y_n \end{bmatrix}_{n\times 1} - \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \cdots & & \cdots \\ \cdots & & \cdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}_{n\times p} \begin{bmatrix} \beta_1 \\ \cdots \\ \cdots \\ \beta_p \end{bmatrix}_{p\times 1} \longrightarrow$$

$$y - X\beta = \begin{bmatrix} y_1 - \sum_{j=1}^{p} x_{1j}\beta_j \\ \cdots \\ \cdots \\ y_n - \sum_{j=1}^{p} x_{nj}\beta_j \end{bmatrix}_{n\times 1} \rightarrow (y - X\beta)^T = \begin{bmatrix} y_1 - \sum_{j=1}^{p} x_{1j}\beta_j & \cdots \end{bmatrix}_{1\times n}$$

$$\rightarrow (y - X\beta)^T (y - X\beta) = \boxed{\sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2} \quad \text{term 1}$$

$$\lambda \beta^T \beta = \boxed{\lambda \sum_{j=1}^{p} \beta_j^2} \quad \text{term 2}$$

So $J(\beta) = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$

$$y^* = \begin{bmatrix} y \\ 0 \end{bmatrix} \quad X^* = \begin{bmatrix} X \\ \lambda I \end{bmatrix}$$

Year:........ Month:........ Day:........

$$\min_\beta \|y^* - X^*\beta\|_2^2 \rightarrow J'(\beta) = (y^* - X^*\beta)^T (y^* - X^*\beta)$$

$$y^* - X^*\beta = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{(n+p)\times 1} - \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{np} \\ \lambda & & 0 \\ 0 & \lambda & \lambda \end{bmatrix}_{(n+p)\times p} \begin{bmatrix} \beta_1 \\ \cdots \\ \cdots \\ \beta_p \end{bmatrix}_{p\times 1} \rightarrow$$

$$y^* - X^*\beta = \begin{bmatrix} y_1 - \sum_{j=1}^{p} X_{1j}\beta_j \\ \cdots \\ y_n - \sum_{j=1}^{p} X_{nj}\beta_j \\ 0 - \lambda\beta_1 \\ \cdots \\ 0 - \lambda\beta_p \end{bmatrix}_{(n+p)\times 1} \rightarrow$$

$$(y^* - X^*\beta)^T (y^* - X^*\beta) = \underbrace{\boxed{\sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2}}_{\text{term 1}} + \underbrace{\boxed{\lambda^2 \sum_{j=1}^{p} \beta_j^2}}_{\text{term 2}}$$

Since the two equations have different Coefficients in the second term, they will undergo separate optimization Processes and their solutions won't be the same.

2 question 2.b

Year:........ Month:........ Day:........

2.b) Lasso $\min_\beta \|y - X\beta\|_2^2 + \lambda\|\beta\|_1$

elastic net $\min_\beta \|y - X\beta\|_2^2 + \lambda(\alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2^2)$

For lasso we have $J(\beta) = \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|_1$

define $\begin{cases} \lambda_1 = \lambda\alpha, & y^* = \begin{bmatrix} y \\ 0 \end{bmatrix} \\ \lambda_2 = \lambda(1-\alpha), & X^* = \begin{bmatrix} X \\ \sqrt{\lambda_2}\, I \end{bmatrix} \end{cases}$

now $\min_\beta \underline{\|y^* - X^*\beta\|_2^2} + \lambda_1\|\beta\|_1$ Lasso

from question 2.a we can conclude that

$\underline{\|y^* - X^*\beta\|_2^2} = \|y - X\beta\|_2^2 + \lambda_2\|\beta\|_2^2$

So the overall equation is going to be

$\|y - X\beta\|_2^2 + \lambda_2\|\beta\|_2^2 + \lambda_1\|\beta\|_1 =$

$\|y - X\beta\|_2^2 + \lambda((1-\alpha)\|\beta\|_2^2 + \alpha\|\beta\|_1)$ Elastic net

[ ]:
```
# supplementary data 1
```

You : please explain me the gradient descent algorithm and how to set it up ChatGPT : Certainly! Gradient descent is an optimization algorithm commonly used in machine learning and optimization problems to minimize a function by iteratively moving in the direction of steepest descent of the function. It's particularly useful for finding the minimum of a cost function in the context of training machine learning models.

Here's a step-by-step explanation of the gradient descent algorithm:

1-Initialization: Start by initializing the parameters (coefficients) of the model to some arbitrary values. This could be zeros or random values.

2-Compute the Gradient: Compute the gradient of the cost function with respect to each parameter. The gradient points in the direction of the steepest increase of the function.

3-Update Parameters: Adjust the parameters in the opposite direction of the gradient to minimize the cost function. This adjustment is made by subtracting a fraction of the gradient (scaled by a learning rate) from the current parameter values.

4-Repeat: Repeat steps 2 and 3 until convergence criteria are met, such as reaching a maximum number of iterations or the change in the cost function becoming sufficiently small.

5-Convergence: Check for convergence by evaluating if the change in the cost function or parameter values is below a predefined threshold.

The key component of gradient descent is the learning rate, denoted by (alpha). This parameter controls the size of the steps taken during optimization. If the learning rate is too small, convergence may be slow, while if it's too large, the algorithm may overshoot the minimum or even diverge.

Here's how to set up gradient descent:

1-Define the Cost Function: Start by defining the cost function you want to minimize. This could be the mean squared error for linear regression, the cross-entropy loss for logistic regression, or any other appropriate function for your problem.

2-Compute the Gradient: Calculate the gradient of the cost function with respect to each parameter using calculus or automatic differentiation libraries if available (such as TensorFlow or PyTorch).

3-Choose a Learning Rate: Select an appropriate learning rate ( ). You may need to experiment with different values to find one that works well for your problem.

4-Initialize Parameters: Initialize the parameters (coefficients) of the model.

5-Update Parameters: Use the gradient computed in step 2 to update the parameters using the update rule: := - * J( ), where represents the parameters, is the learning rate, and J( ) is the gradient of the cost function.

Repeat Until Convergence: Repeat steps 2-5 until the algorithm converges to a minimum, as determined by a convergence criterion.

Gradient descent is a fundamental optimization algorithm used in many machine learning algorithms, including linear regression, logistic regression, neural networks, and more. Understanding gradient descent and its variants is crucial for effectively training machine learning models.

[ ]:
```
# supplementary data 2
```

You : just explain the mathematics of the coordinate descent algorithm for LASSO regression
ChatGPT :

[ ]:

ChatGPT 3.5

ChatGPT

Sure, I'd be happy to explain the mathematics behind the coordinate descent algorithm for LASSO linear regression.

LASSO (Least Absolute Shrinkage and Selection Operator) is a linear regression technique that adds a penalty term to the ordinary least squares objective function. This penalty term encourages sparse solutions by shrinking some coefficients towards zero.

The optimization problem for LASSO can be formulated as:

$$\min_{\beta} \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_1$$

Where:

- $\mathbf{X}$ is the design matrix (features)
- $\mathbf{y}$ is the target vector
- β is the coefficient vector
- λ is the penalty hyperparameter
- $\|\cdot\|_2$ denotes the L2-norm (Euclidean norm)
- $\|\cdot\|_1$ denotes the L1-norm (sum of absolute values)

The cyclic coordinate descent algorithm iteratively optimizes one coordinate (feature) at a time

Message ChatGPT...

ChatGPT 3.5

while fixing all others. Here's how it works:

1. Initialize β to any value (e.g., all zeros).
2. Iterate through each coordinate (feature) j and update β_j by optimizing the objective function with respect to β_j, while fixing all other coordinates.
3. Repeat step 2 until convergence or until reaching the maximum number of iterations.

The update rule for coordinate descent in the context of LASSO is given by the soft-thresholding operator:

$$\beta_j^{new} = S(\arg_j, \frac{1}{n}\sum_{i=1}^{n} x_{ij}(y_i - \sum_{k \neq j} x_{ik}\beta_k), \lambda)$$

Where:

- $\arg_j$ is the j-th column of $\mathbf{X}$
- $S(x, \lambda)$ is the soft-thresholding operator defined as:

$$S(x, \lambda) = \mathrm{sign}(x) \cdot (|x| - \lambda)_+$$

where $(\cdot)_+$ denotes the positive part function, which is zero if x is less than λ, and $x - \lambda$ otherwise.

By cyclically updating each coordinate in this manner, we gradually converge towards the optimal

Message ChatGPT...