به نام خدا

نوید نصیری ۹۸۲۹۵۴۳

سوال ۱)

الف)

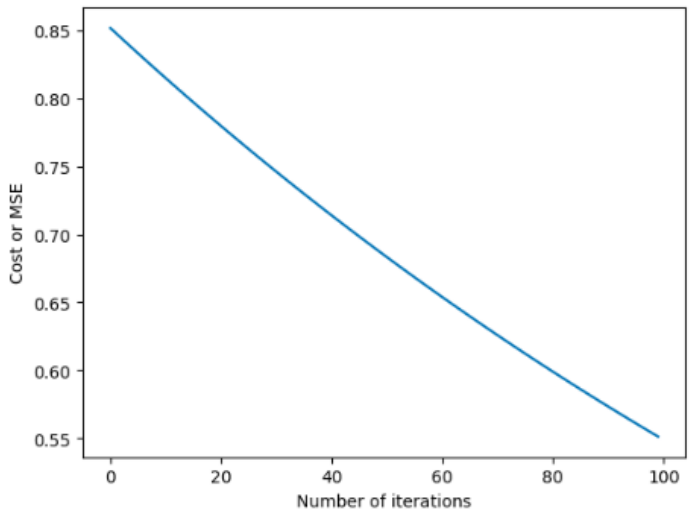
در فایل جوپیتر انجام شد.

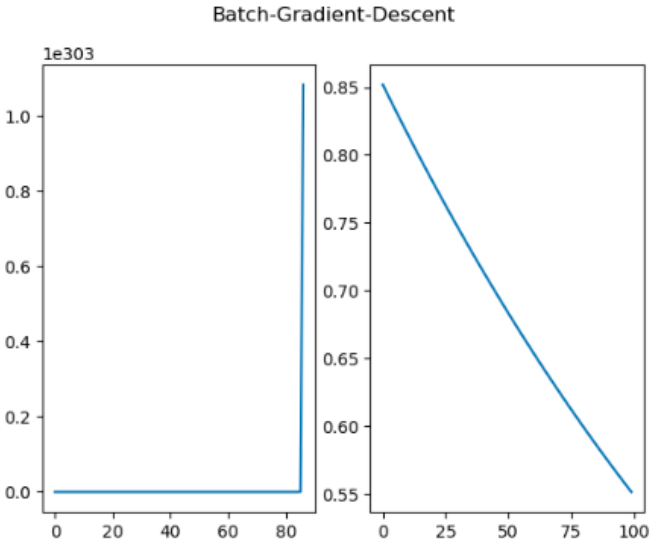
ب , ه)

حالت betch geradient descent :

```
Final Estimate of b and theta :  -8.903970404304659e+171 [-6.64012662e+173 -6.64012662e+173 -6.64012662e+17
Final Estimate of b and theta :  0.7465113127164202 [ 0.82007275 -0.08415376  0.53678293]
betch-gradient-descent with normalized data time : 0.07743549346923828
betch-gradient-descent without normalized data time : 0.10062289237976074
```

Out[386]: [<matplotlib.lines.Line2D at 0x7f96fd4ead00>]

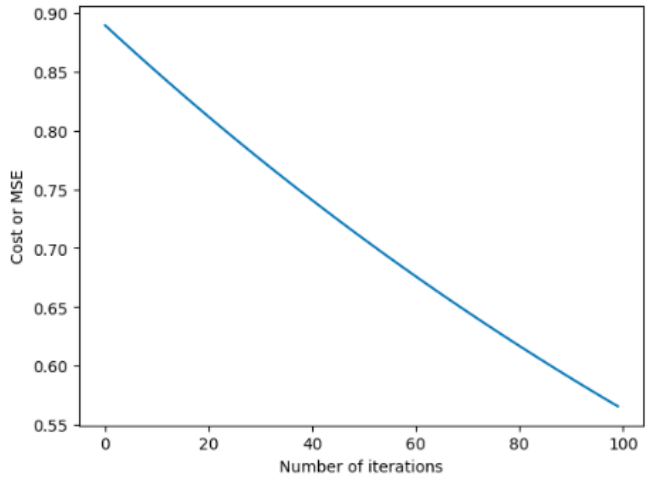




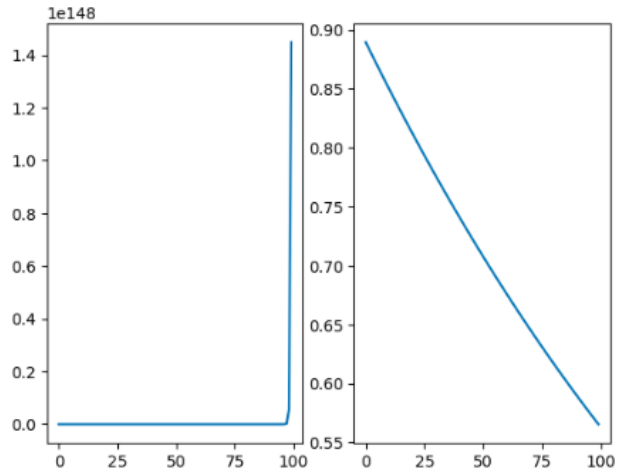
stochastic-gradient descent حالت

```
Final Estimate of b and theta :  -1.662396059525494e+35 [-9.41830488e+36 -9.41830488e+36 -9.418304
Final Estimate of b and theta :  0.083768364334712 [0.37234916 0.95394541 0.6859632 ]
betch-gradient-descent with normalized data time : 0.0946800708770752
betch-gradient-descent without normalized data time : 0.10020613670349121
```

Out[390]: [<matplotlib.lines.Line2D at 0x7f96fd3400d0>]



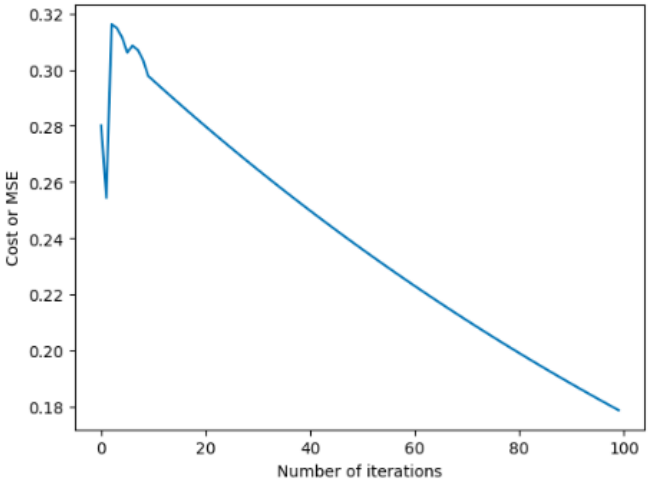
stochastic-Gradient-Descent

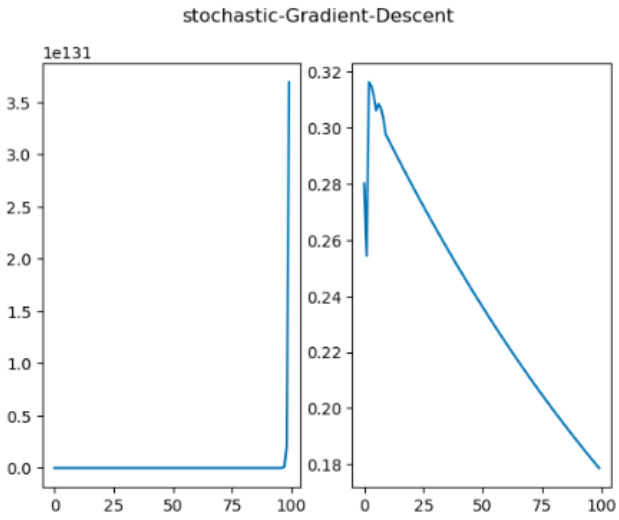


mini-batch-gradient-descent حالت

```
Final Estimate of b and theta :  -6.942927082140315e+62 [-5.01279335e+64 -5.01279335e+64 -5.01279335e+
Final Estimate of b and theta :  0.6488915182541174 [-0.06532214  0.36395947  0.19654602]
betch-gradient-descent with normalized data time : 0.08002829551696777
betch-gradient-descent without normalized data time : 0.09498310089111328
```

Out[394]: [<matplotlib.lines.Line2D at 0x7f96fd1a7cd0>]





تاثیر کوچک بودن نرخ یادگیری :

با کوچکتر بودن نرخ یادگیری، گام های حرکت هر مرحله کوچک تر میشود و بسته به مقدار تعداد ایتریشن هایی که به آن داده شده است، می‌تواند به کاست فانکشن کمتری برسد. اگر این نرخ بسیار کوچک باشد، تعداد ایتریشن های بسیار زیادی نیاز است تا با آن بتوان به مقدار کمینه کاست فانکشن رسید.

تاثیر بزرگ بودن نرخ یادگیری :

با بزرگتر شدن نرخ یادگیری، گام های حرکت در هر مرحله بزرگتر میشود. اگر این مقدار نرخ، بسیار بزرگ باشد ممکن است رفته رفته، از کمینه مقدار کاست فانکشن دور شود و ادامه آن سودی نداشته باشد.

و)

مقادیر بدست آمده با استفاده از کتابخانه sklearn :

```
-0.12830491241144207
[0.2451716  0.33924702 0.55457525]
timer :  0.0023124217987060547
```

ز)

مقادیر بدست آمده با استفاده از روش Normal Equation :

gradient descent with close solution (Normal Equation)

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

n [462]:
```
def normal_equation(X, Y):
    theta = np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T, Y))
    return theta

ne_theta = normal_equation(X_N, Y_N)
print(ne_theta)
```

```
[0.20422842 0.13791416 0.39407646]
```

سوال ۲)

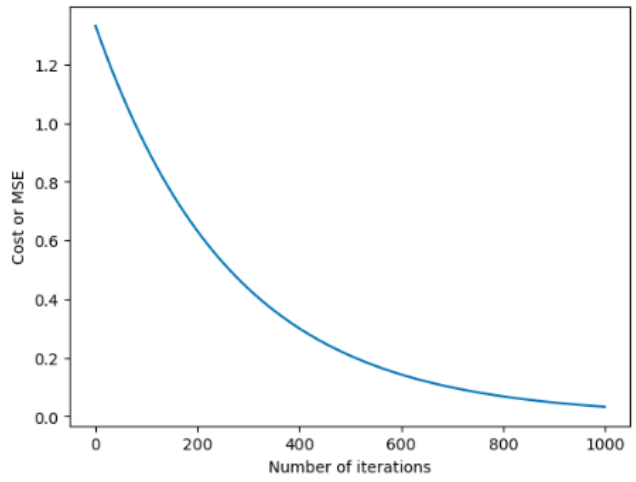
الف) در فایل جوپیتر انجام شد.

ب , ج)

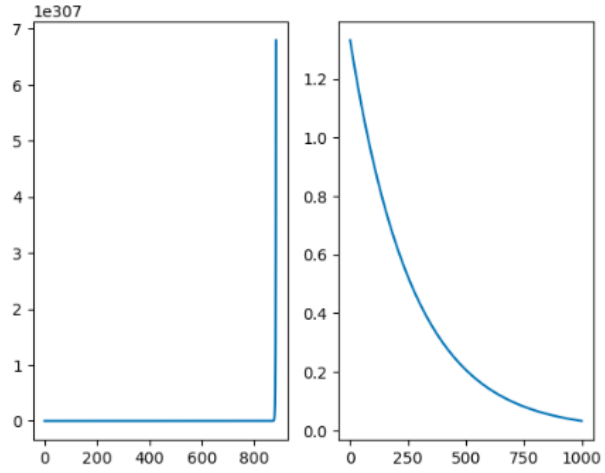
حالتی که از MSE استفاده میکنیم:

```
NOT NORMALIZED DATA :
Final Estimate of b and theta in Gradient Descent with MSE cost function:  9.015948285170586 [346.88317922 347.55201
372 346.76848957]
NORMALIZED DATA :
Final Estimate of b and theta in Gradient Descent with MSE cost function:  0.36885301753915517 [ 0.18444296 -0.14911
187  0.44461762]
betch-gradient-descent with normalized data time : 0.8361392021179199
betch-gradient-descent without normalized data time : 0.8676388263702393
```

```
Out[191]: [<matplotlib.lines.Line2D at 0x7f73a2b25fa0>]
```

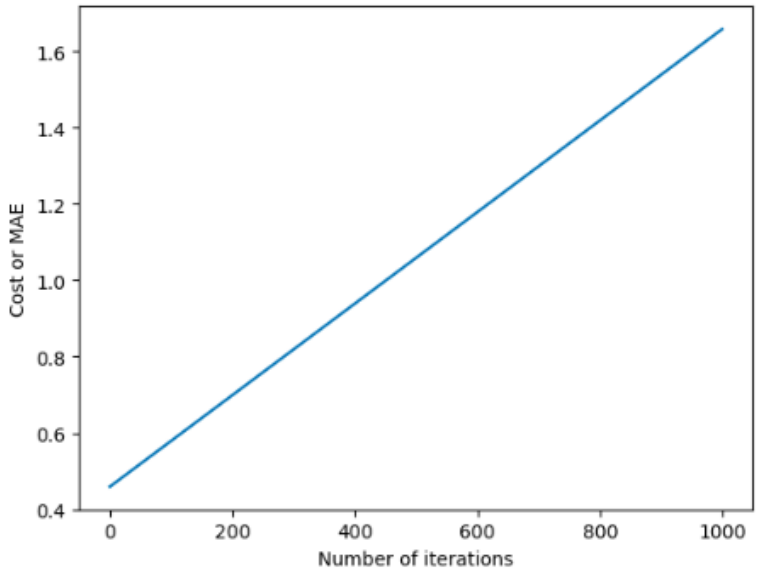


stochastic-Gradient-Descent

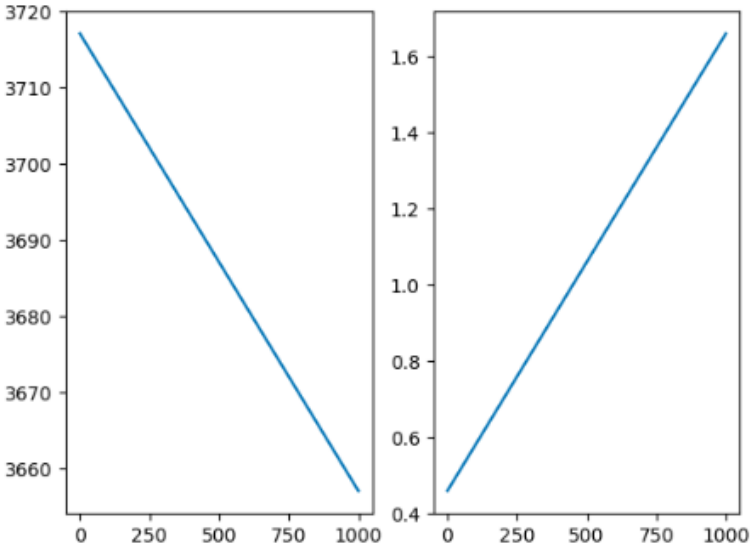


حالتی که از MAE استفاده میکنیم :

```
NOT NORMALIZED DATA :
Final Estimate of theta in Gradient Descent with MAE cost function::  [1.43697898 1.05755063 1.57624352]
NORMALIZED DATA :
Final Estimate of theta in Gradient Descent with MAE cost function::  [1.71487269 1.60193373 1.05734901]
stochastic-gradient-descent with normalized data time : 0.8277714252471924
stochastic-gradient-descent without normalized data time : 0.7898476123809814
```



stochastic-Gradient-Descent



با نرمال شدن داده ها، ضرایب مورد نظر را راحت تر می‌توانیم پیدا کنیم زیرا فراوانی داده ها در یک بازه مشخص قرار می‌گیرد و فاصله زیادی از هم نمی‌گیرند که باعث شود نتایجمان از هم دور شود.

همچنین در حالتی که از MSE استفاده میکنیم، می‌توانیم آن را بیشتر کاهش دهیم و فرایند یادگیری بهتری داشته باشیم. این در حالی است که تابع MAE چون قدرمطلقی است، محدود تر از تابع MSE است و نمی‌تواند به آن اندازه به ما کمک کند و فرایند لرنینگ را به آن اندازه بهبود بخشد. گام هایی که در این روش برداشته می‌شود ثابت است و تنها وابسته به الفا است.

د) و در اخر حالت هایی که از تابع های آماده استفاده میکنیم:

implement Linear Regression with sklearn

```
from sklearn.linear_model import LinearRegression

s_time = time.time()

regressor = LinearRegression()

regressor.fit(X_N, Y_N)

timer = time.time() - s_time

print(regressor.intercept_)

print(regressor.coef_)

print("timer : " , timer)
```

```
-0.12830491241144207
[0.2451716  0.33924702 0.55457525]
timer :  0.0026140213012695312
```

implement stochastic gradient descent with squared_error

```
from sklearn.linear_model import SGDRegressor

## with squared_error cost functoin
sgd_reg_ = SGDRegressor(max_iter=100, penalty=None, eta0=0.1, loss= 'squared_error')

sgd_reg.fit(X_N, Y_N.ravel())

print(sgd_reg.intercept_, sgd_reg.coef_)

## with huber cost function
sgd_reg_ = SGDRegressor(max_iter=100, penalty=None, eta0=0.1, loss= 'huber')

sgd_reg.fit(X_N, Y_N.ravel())

print(sgd_reg.intercept_, sgd_reg.coef_)
```

```
[-0.05777343] [0.23999235 0.19496372 0.02214717]
[-0.04062112] [0.24785941 0.20404076 0.02254694]
```