ECON 425 HW3

January 24, 2024

Due Thu, Feb 1st, 6pm in Bruinlearn

Problem 1

Let $f(x) = x^4 - 6x^2 + 4x + 18$. Use pen and paper to complete (i) and (ii).

(i) Set the initial value $x^{(0)} = 1$ and the learning rate $\alpha = 0.1$. Perform three steps of gradient descent (GD) on f.

(ii) Repeat (i) starting from the point $x^{(0)} = 0$.

Use a computer for the rest of this problem.

(iii) Plot the function f over $x \in [-2.5, 2.5]$. How many local/global minima do you see? What are their approximate values? Can there be other local minima?

(iv) Write your own code for S steps of GD on this function (do not use built-in or third-party GD codes).

(v) Repeat (i) and (ii) using your code with $S = 20$ steps. Do you observe convergence in both cases?

(vi) Now set the learning rate to $\alpha = 0.01$ and repeat (v). Explain why GD performs differently from (v).

Problem 2

The College.csv dataset contains admissions data for a sample of 777 universities. We want to predict the number of applications received (‘Apps’) using the other variables in the dataset.

(i) Let the first 600 observations be the training set and the remaining 177 observations be the test set.

(ii) Fit the OLS regression on the training set, and report the test error obtained.

For the rest of the problem, let the penalization parameter vary on the 1000-point grid from 0.01 to 60.

(iii) Fit the LASSO regression on the training set, with the penalization parameter chosen by 20-fold cross-validation. Report the test error obtained.

(iv) Fit the ridge regression on the training set, with the penalization parameter chosen by leave-one-out cross-validation. Report the test error obtained.

(v) Which of the three models do you prefer? Is there much difference among the test errors?

ML HW 3

January 31, 2024

0.1 Problem 1

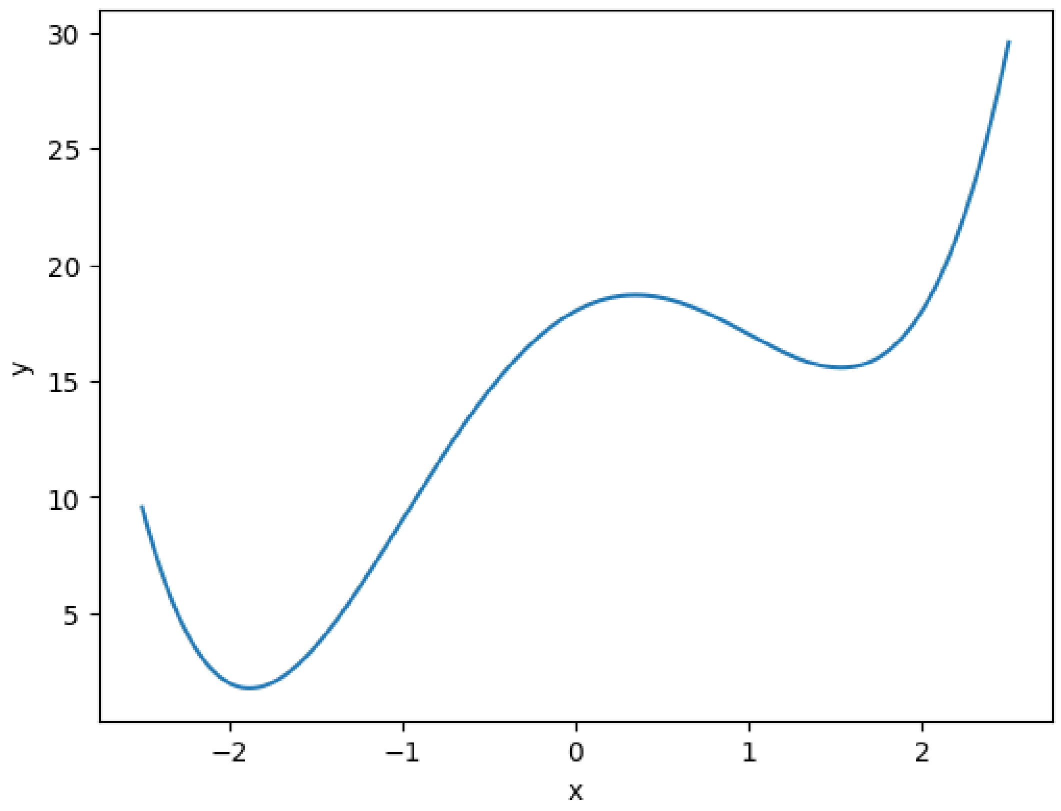
0.1.1 iii)

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Below I plot the function f for x from -2.5 to 2.5. I observe 2 local minima (one around x=-2 with y around 0, one around x=2 with y~15) and 3 local mxima(one at x = -2.5 with y=10, one close to x=0 with y around 19, and one at x = 2.5 with y around 30)

The global minima is next to x= -2 with y~0. The global maxima is at x=2.5 with y around 30.

[2]:
```
x = np.arange(-2.5, 2.51, 0.01)
y = x**4-6*x**2 + 4*x +18
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



0.1.2 iv)

Below I wrote my own code for S steps of GD on the function given in the exercise.

```
[3]: def gd(x, steps):
         xs=[]
         xs.append(x)
         for i in range (19):
             fd = 4 * x**3 - 12*x + 4*x
             x= x - 0.1*fd
             xs.append(x)
         return xs
```

0.1.3 v)

Say S=20 steps. For i) the value of x converges to ~1.4142.

```
[4]: gd(1, 20)
```

```
[4]: [1,
      1.4,
      1.4223999999999999,
      1.4091877474304002,
      1.417186236485252,
      1.4124149508659845,
      1.4152872416220268,
      1.4135673979840149,
      1.4146005525452388,
      1.413981114093114,
      1.4143529396504186,
      1.4141299030385595,
      1.4142637460965501,
      1.4141834478650948,
      1.4142316295388737,
      1.4142027215196682,
      1.4142200666857065,
      1.4142096597137324,
      1.4142159039428652,
      1.4142121574219282]
```

Say S=20 steps. For ii) the value of x converges to 0. Thus, convergence is observed in both cases.

```
[5]: gd(0,20)
```

```
[5]: [0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0]
```

0.1.4 vi)

We start by defining a new function where the learning rate is 0.01.

```
[6]: def gd2(x, steps):
         xs=[]
         xs.append(x)
         for i in range (19):
             fd = 4 * x**3 - 12*x + 4*x
             x= x - 0.01*fd
             xs.append(x)
         return xs
```

Say S=20 steps. For i) the value of x does not converge over the first 20 steps. In this case 20 steps are not enough for convergence because the learning rate is much smaller.

```
[7]: gd2(1,20)
```

```
[7]: [1,
      1.04,
      1.07820544,
      1.1143241590355024,
      1.148123022342038,
      1.179435254567999,
      1.2081631119638494,
      1.2342759577740308,
      1.2578043609956835,
      1.2788312369671284,
      1.2974812343320477,
      1.313909549703096,
      1.3282911671638526,
      1.3408112452155214,
      1.3516570783568256,
      1.36101179656952,
      1.3690497629083402,
      1.3759334955299414,
      1.3818118695581485,
      1.3868193323150038]
```

Say S=20 steps. For i) the value of x does converges to 0.

```
[8]: gd2(0,20)
```

```
[8]: [0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0]
```

0.2 Problem 2

0.2.1 i) + ii)

[9]:
```
df=pd.read_csv("/Users/sandinatatu/Desktop/College.csv")
df['Private'] = df['Private'].replace({'Yes':1, 'No':0})
```

[10]:
```
train=df.iloc[:600]
test=df.iloc[600:]
```

[11]:
```
X_train = train.drop("Apps", axis=1)
y_train = train["Apps"]
X_test = test.drop("Apps", axis=1)
y_test = test["Apps"]
```

[12]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

model = LinearRegression()

model.fit(X_train, y_train)

pred = model.predict(X_test)

res = mean_squared_error(y_test, pred)
```

[13]:
```
print(f"The mean squared error for the test dataset is {res}")
```

```
The mean squared error for the test dataset is 1502077.4348215708
```

0.2.2 iii)

[14]:
```
from sklearn.linear_model import LassoCV

modcv=LassoCV(cv=20)
modcv.fit(X_train, y_train)
```

```
pred = modcv.predict(X_test)

res2=mean_squared_error(y_test, pred)

print(f"The mean squared error for the test dataset is {res2}")
```

```
The mean squared error for the test dataset is 1882607.637374426
```

0.2.3 iv)

[15]:
```
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import LeaveOneOut

loo=LeaveOneOut()

ridgecv= RidgeCV(cv=loo, scoring = 'neg_mean_squared_error')

ridgecv.fit(X_train, y_train)

pred=ridgecv.predict(X_test)

res3 = mean_squared_error(y_test, pred)

print(f"The mean squared error for the test dataset is {res3}")
```

```
The mean squared error for the test dataset is 1502621.0391227088
```

0.2.4 v)

The test error of the OLS linear regression is the lowest, thus it performs better than the other models on unseen data. The test error of the model we obtained using Lasso is the highest, tehrefore out of the 3 models, we would not be using that one.