lab

January 21, 2021

1 Week 6. Optimization. Programming Task

1.0.1 For grading

[1]:
```
#DO NOT CHANGE
import grading
import grading_utils
```

[2]:
```
#DO NOT CHANGE
grader = grading.Grader(assignment_key="Aoii7s9WRQyvONf6kwwjAw",
                        all_parts=["NCrTc", "JigtH", "HlACv"])
```

[18]:
```
# token expires every 30 min
COURSERA_TOKEN = "OAoIh4Q0ahl5nZne"### YOUR TOKEN HERE
COURSERA_EMAIL = "manuelalejandromartinezf@gmail.com"### YOUR EMAIL HERE
```

Let us consider the **House Pricing** dataset, where you have a lot of information about the houses being sold and you aim to produce the price of the house.

Firstly, let us import basic libraries (`numpy` (docs) for matrix operations and `pandas` (docs) for convinient dataset workaround):

[4]:
```
import numpy as np
import pandas as pd
```

1.0.2 Task 1. Reading and Preparing

[5]:
```
datX=np.load('x_train.npy')
datY=np.log(np.load('y_train.npy'))
datX=pd.DataFrame(datX, columns=datX.dtype.names)
datX
```

[5]:
```
             date  bedrooms  bathrooms  sqft_living  sqft_lot  floors  \
0      2014-09-16       5.0       3.25         3710     34200     2.0
1      2014-11-18       3.0       1.75         2820      8879     1.0
2      2014-11-10       3.0       1.00         1240    239144     1.0
```

```
3      2015-04-16        4.0        2.50         2670       8279      2.0
4      2014-07-23        3.0        2.25         2700       4025      2.0
…             …        …        …            …        …       …
14995  2014-05-21        4.0        2.75         2290       6120      2.0
14996  2015-04-01        3.0        2.00         1430       9250      1.0
14997  2014-07-11        2.0        1.00          640       7768      1.0
14998  2014-05-15        3.0        1.00         1630      10304      1.0
14999  2014-11-20        2.0        1.00          720       4592      1.0

       waterfront  condition  grade  sqft_above  sqft_basement  yr_built  \
0           False          3      8        2510           1200      1986
1           False          5      7        1540           1280      1920
2           False          3      6        1240              0      1921
3           False          3      7        2670              0      1999
4           False          4      8        1760            940      1907
…               …        …      …          …             …       …
14995       False          4      7        2170            120      1926
14996       False          4      8         990            440      1983
14997       False          3      6         640              0      1942
14998       False          5      7        1630              0      1953
14999       False          4      6         720              0      1943

       yr_renovated  zipcode        lat        long
0                 0    98074  47.610100 -122.046997
1              1957    98146  47.509399 -122.375999
2              1992    98038  47.430302 -122.045998
3                 0    98148  47.429199 -122.328003
4                 0    98122  47.607399 -122.293999
…                 …       …          …          …
14995             0    98115  47.674599 -122.327003
14996             0    98052  47.695202 -122.096001
14997             0    98106  47.514999 -122.359001
14998             0    98155  47.754799 -122.317001
14999             0    98199  47.653400 -122.403999

[15000 rows x 16 columns]
```

Okay, we manage to load the data (you can read more about the `load` here. But it is not a necessity). We are going to use linear models to work with it, but firstly we need to come up with idea what features should we include in the model at all (which feature the price is lineary dependent on):

Do not forget to install seaborn. You can do that by running `pip install seaborn` in the command line locally, or simply by running the next sell:

[6]:
```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in /opt/conda/lib/python3.7/site-packages
```

```
(0.10.1)
Requirement already satisfied: pandas>=0.22.0 in /opt/conda/lib/python3.7/site-
packages (from seaborn) (1.0.3)
Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.7/site-
packages (from seaborn) (1.18.4)
Requirement already satisfied: matplotlib>=2.1.2 in
/opt/conda/lib/python3.7/site-packages (from seaborn) (3.2.1)
Requirement already satisfied: scipy>=1.0.1 in /opt/conda/lib/python3.7/site-
packages (from seaborn) (1.4.1)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.7/site-
packages (from pandas>=0.22.0->seaborn) (2020.1)
Requirement already satisfied: python-dateutil>=2.6.1 in
/opt/conda/lib/python3.7/site-packages (from pandas>=0.22.0->seaborn) (2.8.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-
packages (from matplotlib>=2.1.2->seaborn) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=2.1.2->seaborn) (1.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=2.1.2->seaborn) (2.4.7)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-
packages (from python-dateutil>=2.6.1->pandas>=0.22.0->seaborn) (1.14.0)
```

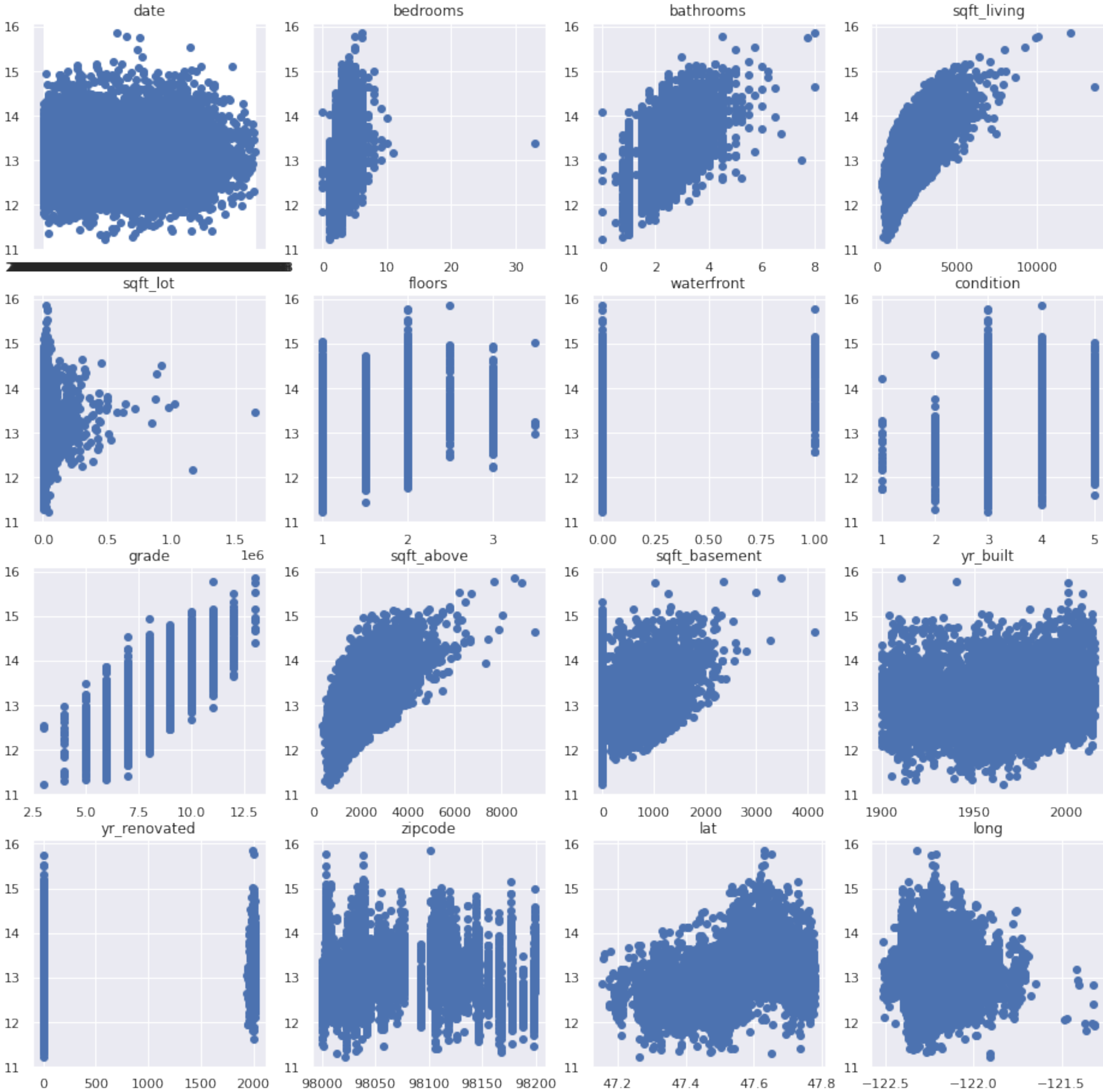
In order to do it let us plot every feature vs the price. Firstly, we import nice plotting modules:

[7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

%matplotlib inline
```

[8]:
```
f, ax=plt.subplots(4, 4, figsize=(16,16))

for i, name in enumerate(datX.columns):
    ax[i//4][i%4].scatter(datX[name], datY)
    ax[i//4][i%4].set_title(name)
```



Let us say, that we choose to work the following set of features: + `bedrooms` + `bathrooms` + `sqft_living` + `floors` + `condition` + `grade` + `sqft_above` + `sqft_basement` + `long` + `lat`

Clear the dataset from all the other features and create: 1. matrix X, all elements should be real numbers 2. number N – number of considered houses 3. number m – number of new features

Hint: it is easier to clean columns from dataset (you should look here for insipration) and the get a matrix with `.values`

Warning: Please use features in the order mentioned above!

[74]:
```
#your code goes here
X=(datX[["bedrooms","bathrooms","sqft_living","floors","condition","grade","sqft_above","sqft_
 ↪values
N=np.shape(X)[0]
```

```
m=np.shape(X)[1]
```

Run the following cells to automatically check results of your code:

[10]:
```
## GRADED PART, DO NOT CHANGE!
grader.set_answer("NCrTc", grading_utils.test_reader(X, N, m))
```

[11]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
You used an invalid email or your token may have expired. Please make sure you
have entered all fields correctly. Try generating a new token if the issue still
persists.
```

Consider that we are interested in the loss of the model we discussed in the video:

- Assume we have input data that is denoted as $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N$
- House prices for this input data are known $y_1, y_2, \ldots, y_N$

We propose a **simple linear model** for this task:

$$\hat{y}_i = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$$

As a loss function we will use the mean squared error (**MSE**):

$$Loss(\vec{w}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

1.0.3 Task 2. Compute analytically the $Loss(\vec{w})$ function

Please, keep the signature of the function and enter the code only under `your code goes here`. **Attention**: try to avoid usage of `for` cycles! The easiest way to do it is by using matrix operations.

Hint: to get nice w_0 coefficient it is convinient to add to the `X` matrix the column of 1 with `np.concatenate` documentation

[12]:
```
def loss(w, X, y):
    #your code goes here
    X=np.concatenate((np.ones(shape=(N,1)),X),axis=1)
    lossValue=np.mean((y-(np.dot(X,w.T)))**2)
    return lossValue
```

Run the following cells to automatically check results of your code:

[13]:
```
## GRADED PART, DO NOT CHANGE!
grader.set_answer("JigtH", grading_utils.test_loss(loss, X, datY))
```

[14]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
You used an invalid email or your token may have expired. Please make sure you
have entered all fields correctly. Try generating a new token if the issue still
persists.
```

1.0.4 Task 3. Compute analyticaly the gradient of the $Loss(\vec{w})$

Please, enter your answer in the cell below (it should be a `markdown` cell). You can either specify each partial derivative $\frac{\partial Loss}{\partial w_i}$ or $\nabla Loss$ altogether using matrix operations.

/ = -2/N Σ (yi-w0-wixi)xi

1.0.5 Task 4. Write a function to compute the gradient of the Loss function in the given point

Please, keep the signature of the function and enter the code only under `your code goes here`. **Attention**: try to avoid usage of `for` cycles! The easiest way to do it is by using matrix operations.

[42]:
```
def grad(w_k, X, y):
    #your code goes here
    N = X.shape[0]
    one = np.ones((X.shape[0], 1))
    Xbar = np.concatenate((one, X), axis = 1)
    return 2/N * (Xbar.T).dot(Xbar.dot(w_k)-y)
```

Run the following cells to automatically check your function.

[20]:
```
## GRADED PART, DO NOT CHANGE!
grader.set_answer("HlACv", grading_utils.test_grad(grad, X, datY))
```

[21]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

1.0.6 Task 5. Write gradient descent

How it is time to formulate the gradient descent! As you remeember, the idea here is that:

$$\vec{w}^{k+1} = \vec{w}^k - \alpha_k \cdot \nabla Loss(\vec{w}^k$$

We propose that you use constant $\alpha_k = \alpha$. Assume that the method should stop in two cases: + if the number of iterations is to high (`maxiter`) + if the length of the gradient is low enough (<`eps`) to call an extremum

Please, keep the signature of the function and enter the code only under `your code goes here`.

[112]:

```
def gradDescent(w_init, alpha, X, y, maxiter=500, eps=1e-2):
    losses=[]
    w=[w_init]

    for it in range(maxiter):
        w_new = w[-1] - alpha*grad(w[-1],X,y)
        if np.linalg.norm(grad(w_new,X,y))/len(w_new) < eps:
            break
        w.append(w_new)
        losses.append(loss(w_new,X,y))
    return losses,w[-1], it
```

Experiment with several alphas and several intial values of weights. To illustrate, provide graphs for the Loss function over iterations in each case (and, optionally, the distance between weigths from one iteration to the next):

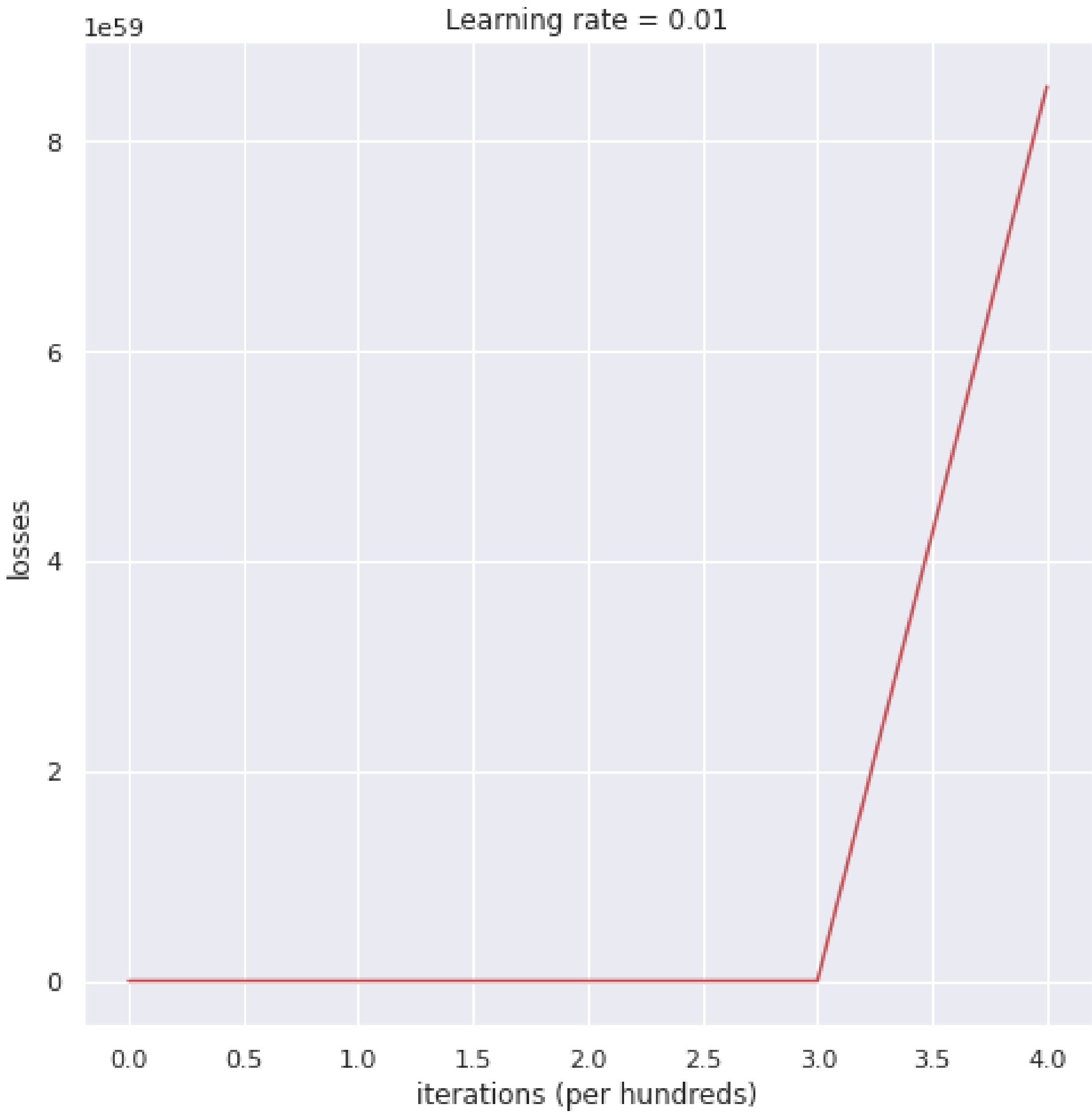
(we provided all key plotting commands for you, but you can always look into this tutorial)

[121]:

```
plt.figure(figsize=(8,8))

#code goes here
w = np.array([1,1,1,1,1,1,1,1,1,1,1])
losses, w,it = gradDescent(w.T,0.01, X, datY,5)
print(losses)
plt.plot( np.arange(it+1),losses, 'r')
plt.ylabel('losses')
plt.xlabel('iterations (per hundreds)')
plt.title("Learning rate = 0.01")
plt.show()
```

```
[6.760047768595087e+17, 2.264243682946674e+28, 7.584008047014016e+38,
2.540237994948272e+49, 8.508441751388078e+59]
```



Let us check the adequacy of the model we created.

Choose several (no less then five) houses (inputs in your `X` matrix) and calculte predicted prices by:

$$\hat{y}_i = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$$

[122]:
```
test = X[:5]
one = np.ones((test.shape[0], 1))
Xbar = np.concatenate((one, test), axis = 1)

y_hat = Xbar.dot(w)
y_hat = np.array(y_hat)
```

```
np.all((y_hat-datY[:5] < 0.001))
```

[122]: True

Compare predicted values with an actual answer (stored in your `y` array). Is it satisfying enough?

1.0.7 Task 6. Discussion

Answer following questions: 1. Does your method converge at least for some alpha? If not, what might be the workaround? 2. How does changing of the alpha influence the speed of convergence? 3. Are the optimal weights in all convergent cases the same? 4. How does this affect the Loss function?