Artificial Neural Networks

Prof. Dr. Sen Cheng

Oct 21, 2019

Problem Set 3: Regression

Tutors: Sandhiya Vijayabaskaran (sandhiya.vijayabaskaran@rub.de), Mohammad M. Nejad (m.mohagheghi@ini.rub.de)

Further Reading: publications etc, book chapters

1. **Analytical solution of linear regression**

   Although in real-world problems you do not know the function that has generated the dataset, here we assume that we know the generating function known as "ground-truth". You will use the functions here to generate data and later fit a linear regression to the data, so you can see how close the regression is to the ground-truth.

   Let us start with the function

   $$y = 4x + 5 + \varepsilon \tag{1}$$

   as the ground-truth to generate a dataset. ε is a uniform random variable, which generates random numbers between -0.5 and 0.5.

   (a) Generate a vector x containing 100 linearly spaced numbers from -3 to 3 (use `numpy.linspace`) and a vector of 100 numbers for ε (use `numpy.random.uniform`). Now compute vector y according to Eq. 1 and make a scatter plot of y against x.

   (b) Use the solution for linear regression to estimate the parameters m and b from the data. Implement the equations yourself using only elementary operations.

   (c) Use the analytical solution for multiple linear regression to estimate the parameters. Make sure you have designed your X matrix correctly. Implement the equations yourself using only elementary matrix operations.

   (d) Compare the parameters you estimated to the ground-truth. Check how good the estimate is by plotting the the regression line along with the data points. Also compute the explained variance, and generate the residual plot.

   (e) Use the scikit-learn python package to perform linear regression. From `sklearn` import `linear_model` and use the method `LinearRegression`. Using the data you generated in the first step, fit a linear regressor. First look at the instructions provided by `help(linear_model.LinearRegression)`. Calculate the explained variance R^2 using `sklearn.metrics.r2_score`.

2. **Polynomial regression**

   Take a stochastic polynomial function

   $$y = -x^5 + 1.5x^3 - 2x^2 + 4x + 3 + \varepsilon, \tag{2}$$

   where $-2 < x < 2$ and ε is a random number uniformly distributed between -1 and 1.

   Follow the steps below:

   (a) Generate 100 data points using the polynomial function

   (b) Fit a 5th-order polynomial with intercept to the dataset. Use the equation for multiple linear regression and a properly designed matrix X to find the coefficients of the polynomial.

(c) Compare the parameters you estimated to the ground-truth. Check how the estimated parameters vary with the number of data points by plotting the error in the estimates versus the number of data points N. Use the l_2-norm of the difference between true and estimated parameters as an error measure, and vary N in steps of 5 up to a range of 100.

3. **Incremental version of linear regression**

Often in practice, datasets can be extremely large, which makes it infeasible to use the analytical solution. Use stochastic gradient descent (SGD) to compute the regression parameters as follows :

(a) Program stochastic gradient descent update for linear regression using the l_2-loss function. Again, only use elementary operations. (Hint: don't forget to shuffle the data before iterating through it!)

(b) Test this incremental form on the function you used in exercise 1.

(c) Compare the parameters you estimated to the ground-truth. Plot both the data and predicted values for a visual check. Check how the number of data points affect the estimates by generating a plot as in Problem 2c. Use a constant learning rate of 0.01 for this and use 3 runs of SGD. Next, keep the number of data points constant at 100 and use 3 runs of SGD, and vary the learning rate in steps of 0.001 up to a range of 0.4. Generate a similar plot of error vs. learning rate.

Solutions

1. **Analytical solution of linear regression**

   (a) Scatter plot of y against x (Fig. 1)

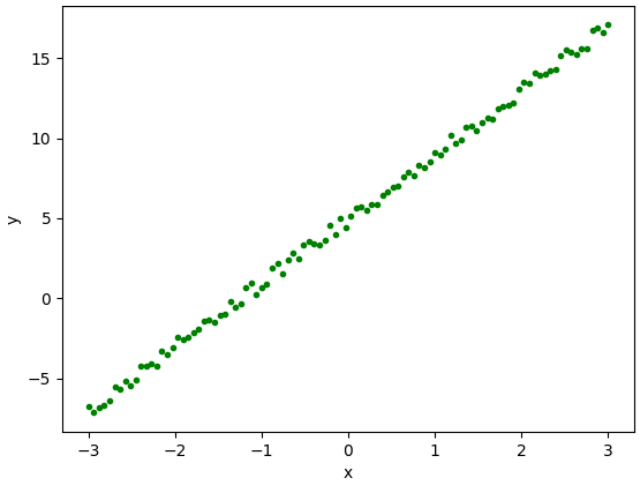
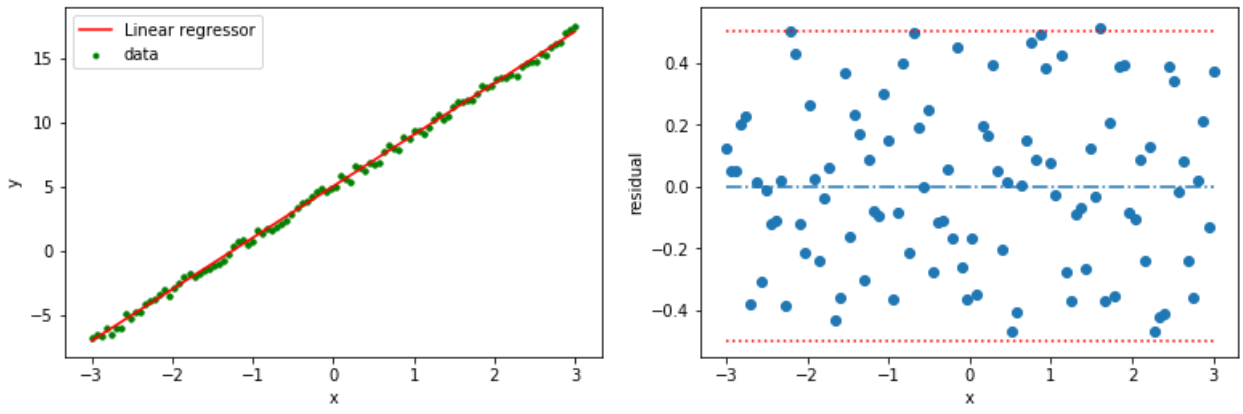


Figure 1: Data

   (b) The estimation of parameters with 4 floating point precision are: $m = 4.0057$ and $b = 4.9547$. You will most probably get different estimates due to the difference in the random number generator used to generate ε. In other words, you will get a different y vector due to a different ε.

   (c) Parameters estimated using the analytical solution for multiple linear regression: $m = 4.0057$ and $b = 4.9547$. The parameters estimated in this part must be identical to solution 1b, when the dataset is identical.

   (d) Plotting regression line together with the data (Fig. 2a) and its corresponding residual plot (Fig. 2b). Explained variance (R2) is 0.998353.



(a) (b)

Figure 2: (a) Data and the linear regressor. (b) The residual plot of the linear regressor.

   (e) The estimated parameters for multiple linear regression using sklearn: $m = 4.0057$ and $b = 4.9547$ and R^2=0.998353.

```
import numpy as np
import matplotlib.pyplot as plt

#.....Solution 1a.....#
size = 100
true_params = [5,4]
x = np.linspace(−3, 3, size)
eps = np.random.uniform(−0.5, 0.5, size) #noise
Y = true_params[0] + true_params[1]∗x + eps

fig, ax = plt.subplots()
ax.scatter(x, Y, c='g', marker='.')
ax.set_xlabel('x')
ax.set_ylabel('y')
fig.savefig('1_a.png', format='png')

#.....Solution 1b.....#
m = (np.sum(x∗Y) − np.sum(x)∗np.sum(Y) / size) / (np.sum(x∗x) −
      ((np.sum(x))∗∗2) / size)
b = np.mean(Y) − m∗np.mean(x)

print('Estimated intercept is %.4f and slope is %.4f' %(b,m))

#.....Solution 1c.....#
bias = np.ones(size)
X = np.vstack((bias, x)).T #design matrix

params = np.linalg.pinv((np.dot(X.T,X)))
params = np.dot(params,X.T)
params = np.dot(params,Y)

print('Estimated intercept is %.4f and slope is %.4f' %tuple(params))

#.....Solution 1d.....#
#Plot data with regression line
fig, ax = plt.subplots()

ax.scatter(x, Y, c='g', marker='.')
ax.set_xlabel('x')
ax.set_ylabel('y')

y_pred = params[0] + params[1] ∗ x

ax.scatter(x, Y, c='g', marker='.', label='data')
ax.plot(x, y_pred, c='r', label='Linear regressor')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('')
ax.legend()
fig.savefig('1_d.png', format='png')

residues = y_pred − Y
```

```
y_mean = np.mean(Y)
ss_data = np.sum((Y − y_mean)**2)
ss_reg= np.sum((y_pred − y_mean)**2)
exp_variance = ss_reg / ss_data
print('Explained variance is = %f' %exp_variance)
#Generate residual plot
fig, ax = plt.subplots()

ax.scatter(x,residues)
ax.plot(x,np.zeros(len(x)), linestyle='−.')
ax.plot(x, 0.5*np.ones(len(x)),c="r", linestyle=':')
ax.plot(x,−0.5*np.ones(len(x)),c="r", linestyle=':')
ax.set_xlabel('x')
ax.set_ylabel('residual')
fig.savefig('1_d2.png', format='png')

#.....Solution 1e.....#

from sklearn import linear_model
from sklearn.metrics import r2_score

lin_reg = linear_model.LinearRegression(fit_intercept=True)
lin_reg.fit(x.reshape(−1,1), Y)
slope = lin_reg.coef_
intercept = lin_reg.intercept_
y_pred_sk = lin_reg.predict(x.reshape(−1,1))
r2_sk = r2_score(Y,y_pred_sk)
print('Estimated intercept is %.4f and slope is %.4f' %(intercept, slope))
print('R2−score (using sklearn) = %f' %r2_sk)
```

2. **Polynomial regression**

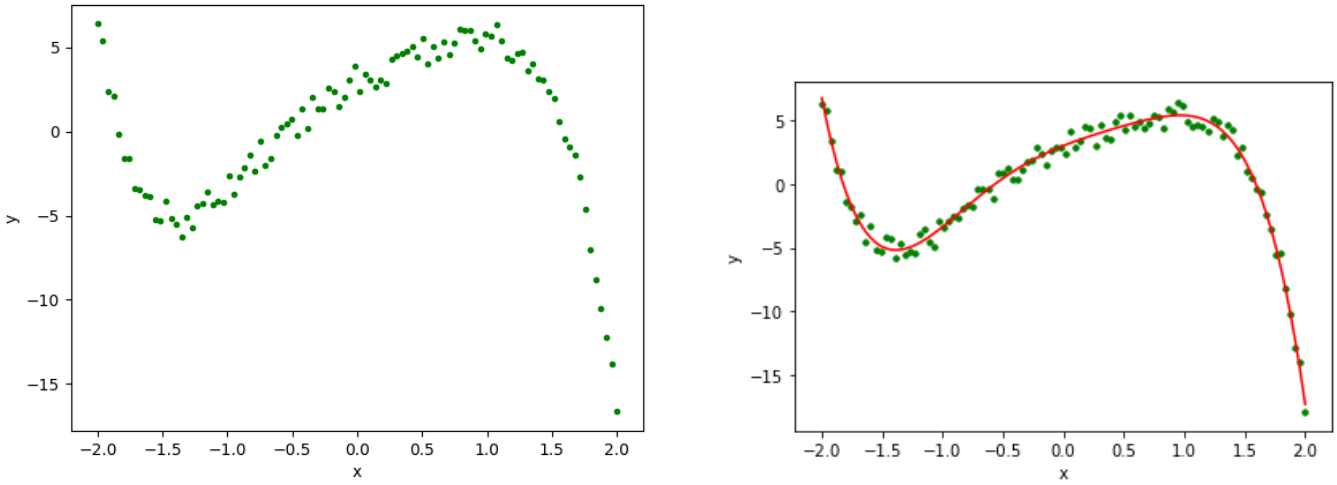
   (a) Scatter plot of y against x (Fig. 3a)

   (b) Applying multiple linear regression to the data generated using the polynomial function (Fig. 3b).

   (c) Increasing the number of data points generally reduces the l_2-norm of the difference between the true and estimated parameters (Fig. 4).

```
import numpy as np
import matplotlib.pyplot as plt

#.....Solution 2a.....#
def generate_data(size) :
    x = np.linspace(−2,2,size)
    eps = np.random.uniform(−1,1,size)
    true_params = [3,4,−2,1.5,0,−1]
    Y = (true_params[0] + true_params[1]*x + true_params[2]*(x**2) +
        true_params[3]*(x**3) + true_params[4]*x**4 + true_params[5]*x**5) + eps
```



(a) (b)

Figure 3: (a) Scatter plot of the data (b) Data and the linear regressor

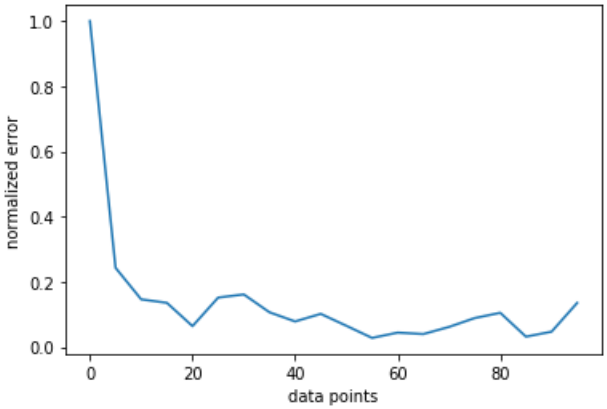


Figure 4: The role of the number of data point on the normalized error.

```
    bias = np.ones(size)
    X = np.vstack((bias,x,x**2,x**3,x**4,x**5)).T #design matrix

    return x,X,Y

size = 100
x, X, Y = generate_data(size)
fig, ax = plt.subplots()
ax.scatter(x, Y, c='g', marker='.')
ax.set_xlabel('x')
ax.set_ylabel('y')
fig.savefig('2_a.png', format='png')

#.....Solution 2b.....#
params = np.linalg.pinv((np.dot(X.T,X)))
params = np.dot(params,X.T)
params = np.dot(params,Y)

fig, ax = plt.subplots()
ax.scatter(x, Y, c='g', marker='.')
```

```
y_pred = np.dot(params, X.T)
#OR y_pred = (params[0] + params[1]*x + params[2]*(x**2) +
# params[3]*(x**3) + params[4]*x**4 + params[5]*x**5)
ax.scatter(x, Y, c='g', marker='.')
ax.plot(x, y_pred, c='r', linewidth=2)
ax.set_xlabel('x')
ax.set_ylabel('y')
fig.savefig('2_b.png', format='png')

#.....Solution 2c.....#
error = []
N = np.arange(0,100,5)
true_params = [3,4,-2,1.5,0,-1]
for n in N :
    x,X,Y = generate_data(n)
    params = np.linalg.pinv((np.dot(X.T,X)))
    params = np.dot(params,X.T)
    params = np.dot(params,Y)
    err = np.linalg.norm(true_params - params, 2)/np.linalg.norm(true_params,2)
    error.append(err)

fig, ax = plt.subplots()
ax.plot(N,error,linewidth=2)
ax.set_xlabel('data points')
ax.set_ylabel('normalized error')
fig.savefig('2_c.png', format='png')
```

3. **Incremental form of regression**

   (a) For the implementation look at the `sgd_update` function in the python code below.

   (b) Test the above function on data from Problem 1.

   (c) Plotting the data points and the linear regression of the data (Fig. 5). Increasing the number of data point decreases the error (Fig. 6a). In general, smaller learning rates lead to a lower error when compared to larger learning rates. However, when the learning rate is too small and the number of iterations is limited ($N = 100$), the gradient descent does not reach the minimum and the error can be large (Fig. 6b). Thus, the choice of learning rate has to be made after careful consideration.

```
import numpy as np
import matplotlib.pyplot as plt

#.....Solution 3a.....#
def sgd_update(X,Y,lr=0.001,runs=1):
    ''' X : X matrix like in examples 1 and 2
        Y : observed values of dependent variable
        lr : learning rate
        runs : number of iterations of stochastic gradient descent
    '''
```

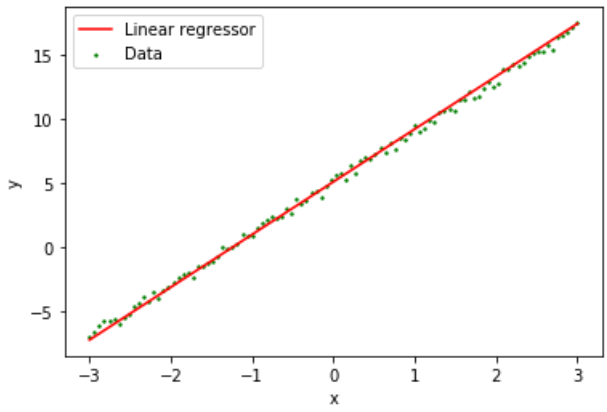
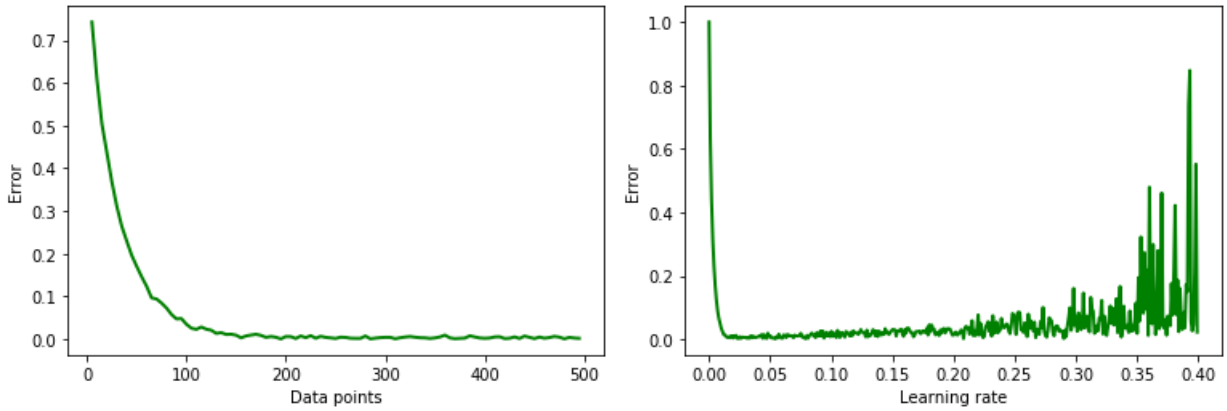


Figure 5: (a) Data and the linear regressor.



(a) (b)

Figure 6: The impact of the number of data points (a) and the learning rate (b; $N = 100$) on the error.

```
    theta = np.zeros(len(X[0,:]) )
    for j in range(runs):
        p = np.random.permutation(len(X))
        X = X[p]
        Y = Y[p]
        for i in range(len(X)) :
            theta = theta + np.dot(lr∗(Y[i] − np.dot(X[i], theta)), X[i]) #eq. 22
    return theta

#.....Solution 3b.....#
size = 100
true_params = [5,4]
x = np.linspace(−3, 3, size)
eps = np.random.uniform(−0.5, 0.5, size) #noise
Y = true_params[0] + true_params[1]∗x + eps
bias = np.ones(100)
X = np.vstack((bias, x)).T

theta = sgd_update(X, Y, 0.1, 5)

#.....Solution 3c.....#
y_pred = theta[0] + theta[1] ∗ x
```

```
fig, ax = plt.subplots()
ax.scatter(x, Y, s=8, c='g', marker='.',label='Data')
ax.plot(x, y_pred, c='r',label='Linear regressor')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()
# effect of data points
error_n = []
N = np.arange(5,500,5)
for n in N :
    x = np.linspace(−3, 3, n)
    eps = np.random.uniform(−0.5, 0.5, n) #noise
    Y = true_params[0] + true_params[1]∗x + eps
    bias = np.ones(n)
    X = np.vstack((bias, x)).T
    params = sgd_update(X,Y,0.01,3)

    err = np.linalg.norm(true_params − params, 2)/np.linalg.norm(true_params,2)
    error_n.append(err)

fig, ax = plt.subplots()
ax.plot(N, error_n, linewidth=2, c='g')
ax.set_xlabel('Data points')
ax.set_ylabel('Error')
# effect of learning rate
error_lr = []
lr = np.arange(0,0.4,0.001)

x = np.linspace(−3, 3, 100)
eps = np.random.uniform(−0.5, 0.5, 100) #noise
Y = true_params[0] + true_params[1]∗x + eps
bias = np.ones(100)
X = np.vstack((bias, x)).T

for l in lr :
    params = sgd_update(X,Y,l,3)
    err = np.linalg.norm(true_params − params, 2)/np.linalg.norm(true_params,2)
    error_lr.append(err)

fig, ax = plt.subplots()
ax.plot(lr, error_lr, linewidth=2, c='g')
ax.set_xlabel('Learning rate')
ax.set_ylabel('Error')
```