linear_regression

January 18, 2021

0.1 Linear regression workbook

This workbook will walk you through a linear regression example. It will provide familiarity with Jupyter Notebook and Python. Please print (to pdf) a completed version of this workbook for submission with HW #1.

ECE C147/C247, Winter Quarter 2021, Prof. J.C. Kao, TAs: N. Evirgen, A. Ghosh, S. Mathur, T. Monsoor, G. Zhao

[ ]:

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

#allows matlab plots to be generated in line
%matplotlib inline
```

0.1.1 Data generation

For any example, we first have to generate some appropriate data to use. The following cell generates data according to the model: $y = x - 2x^2 + x^3 + \epsilon$

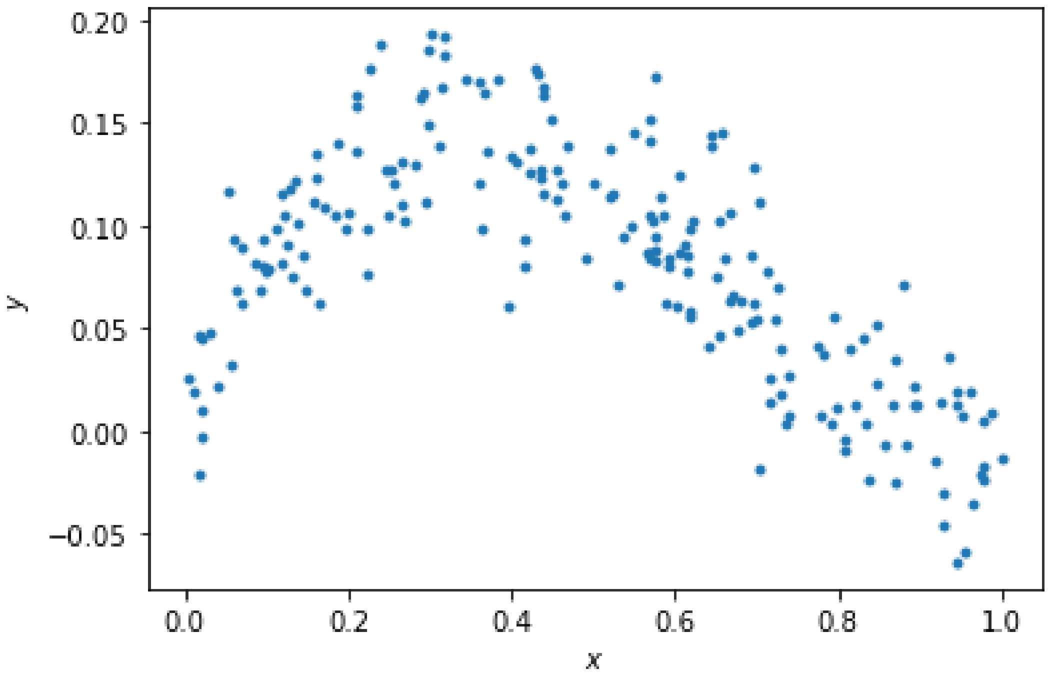
[ ]:

```
np.random.seed(0)  # Sets the random seed.
num_train = 200     # Number of training data points

# Generate the training data
x = np.random.uniform(low=0, high=1, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

[ ]:

```
Text(0, 0.5, '$y$')
```



0.1.2 QUESTIONS:

Write your answers in the markdown cell below this one:

(1) What is the generating distribution of x?

(2) What is the distribution of the additive noise ϵ?

0.1.3 ANSWERS:

(1) The distrubution of x is uniform distribution.

(2) The distrubution of ϵ is normal distrubution.

0.1.4 Fitting data to the model (5 points)

Here, we'll do linear regression to fit the parameters of a model $y = ax + b$.

[ ]:

```
# xhat = (x, 1)
xhat = np.vstack((x, np.ones_like(x)))

# ==================== #
# START YOUR CODE HERE #
# ==================== #
# GOAL: create a variable theta; theta is a numpy array whose elements are [a, b]
# theta = np.zeros(2) # please modify this line
theta = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))

# ================== #
# END YOUR CODE HERE #
# ================== #
```

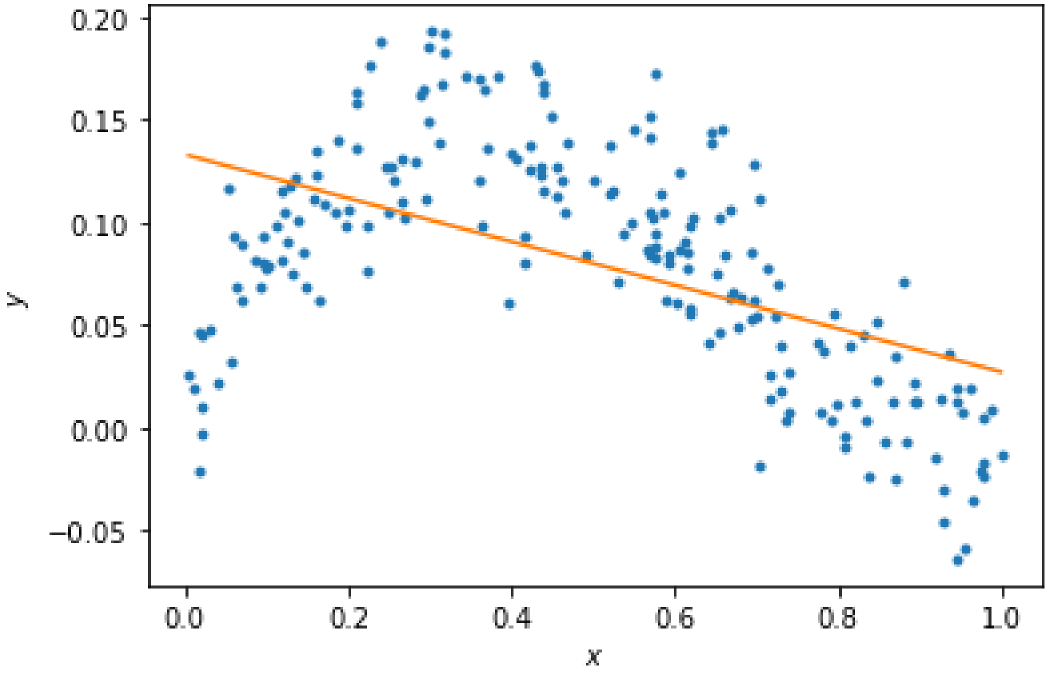
[ ]:

```
# Plot the data and your model fit.
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')

# Plot the regression line
xs = np.linspace(min(x), max(x),50)
xs = np.vstack((xs, np.ones_like(xs)))
plt.plot(xs[0,:], theta.dot(xs))
```

[ ]:

```
[<matplotlib.lines.Line2D at 0x12cda468370>]
```



0.1.5 QUESTIONS

(1) Does the linear model under- or overfit the data?

(2) How to change the model to improve the fitting?

0.1.6 ANSWERS

(1) The linear model underfits the data.

(2) Increasing the order of the model can help improving the fitting.

0.1.7 Fitting data to the model (10 points)

Here, we'll now do regression to polynomial models of orders 1 to 5. Note, the order 1 model is the linear model you prior fit.

[ ]:

```
N = 5
xhats = []
thetas = []

# ===================== #
# START YOUR CODE HERE #
# ===================== #

# GOAL: create a variable thetas.
# thetas is a list, where theta[i] are the model parameters for the polynomial fit of order i+1.
#   i.e., thetas[0] is equivalent to theta above.
#   i.e., thetas[1] should be a length 3 np.array with the coefficients of the x^2, x, and 1 respectively.
#   ... etc.

xhat = np.vstack((x, np.ones_like(x)))
theta = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))
xhats.append(xhat)
thetas.append(theta)

for i in range(1, N):
    cur = np.power(x, i+1)
    xhat = np.vstack((cur, xhat))
    theta = np.linalg.inv(xhat.dot(xhat.T)).dot(xhat.dot(y))
    xhats.append(xhat)
    thetas.append(theta)


# =================== #
# END YOUR CODE HERE #
# =================== #
```

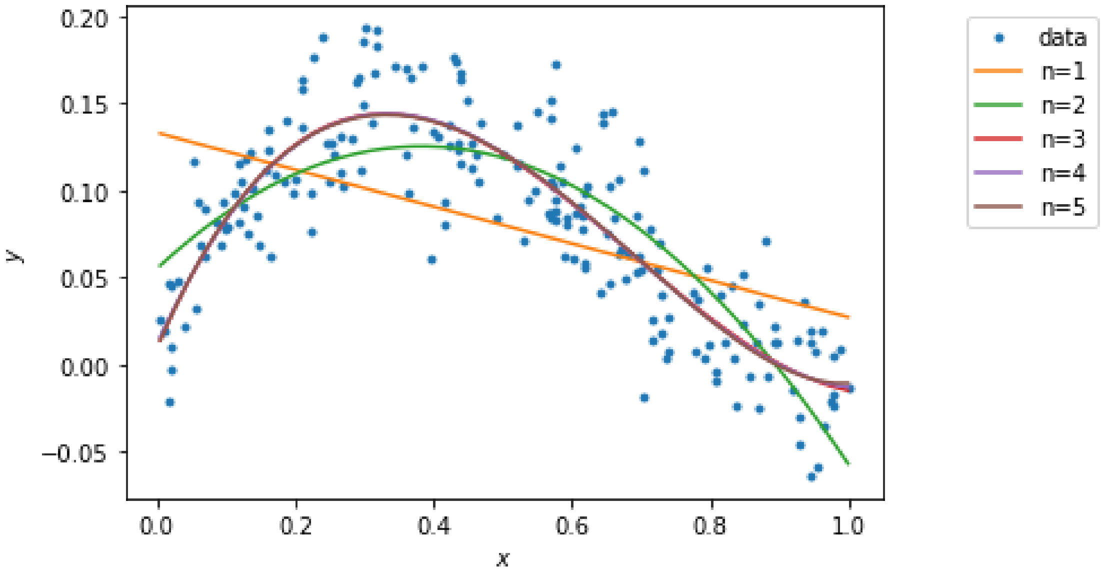
[ ]:

```
# Plot the data
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')

# Plot the regression lines
plot_xs = []
for i in np.arange(N):
    if i == 0:
        plot_x = np.vstack((np.linspace(min(x), max(x),50), np.ones(50)))
    else:
        plot_x = np.vstack((plot_x[-2]**(i+1), plot_x))
    plot_xs.append(plot_x)

for i in np.arange(N):
    ax.plot(plot_xs[i][-2,:], thetas[i].dot(plot_xs[i]))

labels = ['data']
[labels.append('n={}'.format(i+1)) for i in np.arange(N)]
bbox_to_anchor=(1.3, 1)
lgd = ax.legend(labels, bbox_to_anchor=bbox_to_anchor)
```



0.1.8 Calculating the training error (10 points)

Here, we'll now calculate the training error of polynomial models of orders 1 to 5.

[ ]:

```
training_errors = []

# ==================== #
# START YOUR CODE HERE #
# ==================== #
# GOAL: create a variable training_errors, a list of 5 elements,
# where training_errors[i] are the training loss for the polynomial fit of␣
 ↪order i+1.
y_pred = []
for i in range(0, N):
    y_pred.append(thetas[i].dot(xhats[i]))
    training_errors.append(np.sum(np.power((y-y_pred[i]), 2))/2)
# ================== #
# END YOUR CODE HERE #
# ================== #

print ('Training errors are: \n', training_errors)
```

```
Training errors are:
 [0.2379961088362701, 0.10924922209268528, 0.08169603801105371,
0.0816535373529698, 0.08161479195525292]
```

0.1.9 QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

0.1.10 ANSWERS

(1) the polynomial with order 5 has the lowest training error

(2) The higher order polyminals will always fit the training data better than lower order ones becuaue the higher the order is, the more freedom the function has to fit the data. Whether the function will overfit the data is another question though.

0.1.11 Generating new samples and testing error (5 points)

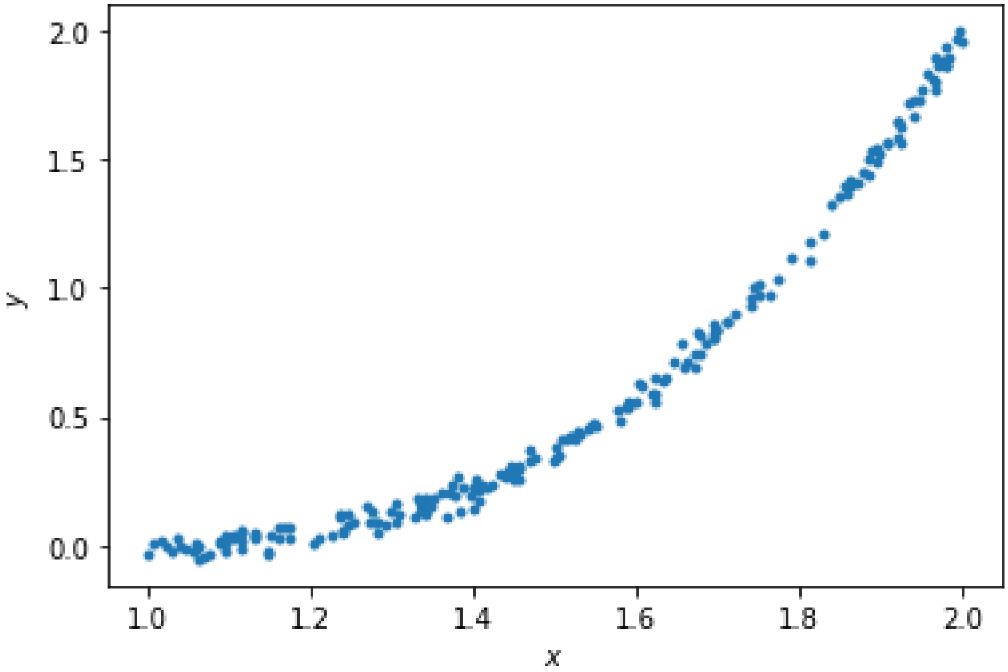
Here, we'll now generate new samples and calculate testing error of polynomial models of orders 1 to 5.

[ ]:

```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

[ ]:

```
Text(0, 0.5, '$y$')
```



```
[ ]: xhats = []
     for i in np.arange(N):
         if i == 0:
             xhat = np.vstack((x, np.ones_like(x)))
             plot_x = np.vstack((np.linspace(min(x), max(x),50), np.ones(50)))
         else:
             xhat = np.vstack((x**(i+1), xhat))
             plot_x = np.vstack((plot_x[-2]**(i+1), plot_x))

         xhats.append(xhat)
```

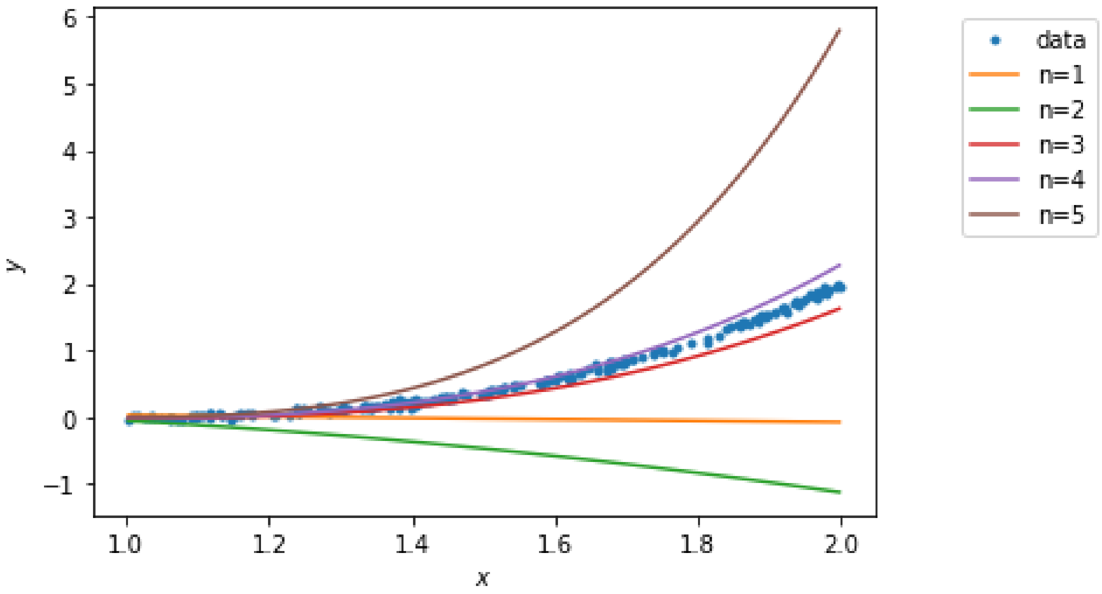
```
[ ]: # Plot the data
     f = plt.figure()
     ax = f.gca()
     ax.plot(x, y, '.')
     ax.set_xlabel('$x$')
     ax.set_ylabel('$y$')

     # Plot the regression lines
     plot_xs = []
     for i in np.arange(N):
         if i == 0:
             plot_x = np.vstack((np.linspace(min(x), max(x),50), np.ones(50)))
         else:
```

```
        plot_x = np.vstack((plot_x[-2]**(i+1), plot_x))
    plot_xs.append(plot_x)

for i in np.arange(N):
    ax.plot(plot_xs[i][-2,:], thetas[i].dot(plot_xs[i]))

labels = ['data']
[labels.append('n={}'.format(i+1)) for i in np.arange(N)]
bbox_to_anchor=(1.3, 1)
lgd = ax.legend(labels, bbox_to_anchor=bbox_to_anchor)
```



[ ]:

```
testing_errors = []

# ==================== #
# START YOUR CODE HERE #
# ==================== #

# GOAL: create a variable testing_errors, a list of 5 elements,
# where testing_errors[i] are the testing loss for the polynomial fit of order␣
 ↪i+1.
y_test = []
for i in range(0, N):
    y_test.append(thetas[i].dot(xhats[i]))
    testing_errors.append(np.sum(np.power((y-y_test[i]), 2))/2)

# ================== #
# END YOUR CODE HERE #
# ================== #

print ('Testing errors are: \n', testing_errors)
```

```
Testing errors are:
 [80.8616518455059, 213.19192445058022, 3.1256971082744753, 1.1870765189429007,
214.91021834596475]
```

0.1.12 QUESTIONS

(1) What polynomial has the best testing error?

(2) Why polynomial models of orders 5 does not generalize well?

0.1.13 ANSWERS

(1) The polynomial with order 4 has the lowest testing error.

(2) The polynomial models of orders 5 overfits the data.