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 2020, Prof. J.C. Kao, TAs W. Feng, J. Lee, K. Liang, M. Kleinman, C. Zheng

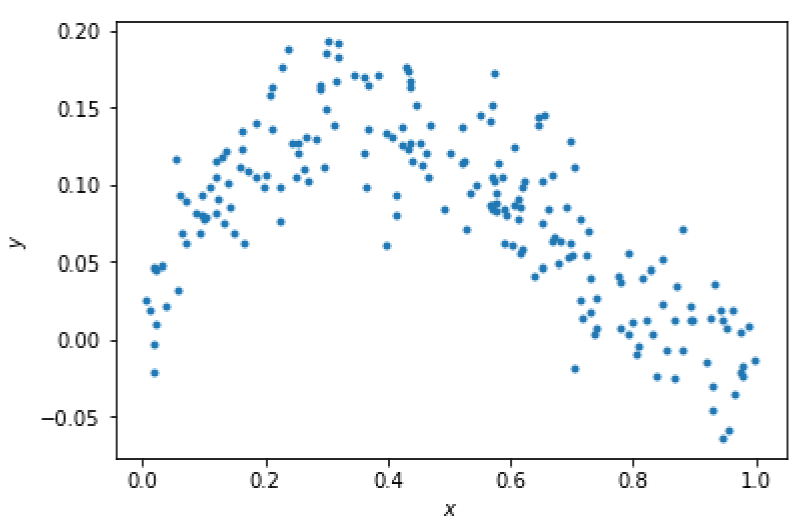
In [88]:
```
import numpy as np
import matplotlib.pyplot as plt

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

Data generation

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

In [89]:
```
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('$y$')
```

Out[89]: `Text(0,0.5,'$y$')`



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?

ANSWERS:

(1) $x \sim Uniform[0, 1]$

(2) $\epsilon \sim \mathcal{N}(0, 0.03^2)$

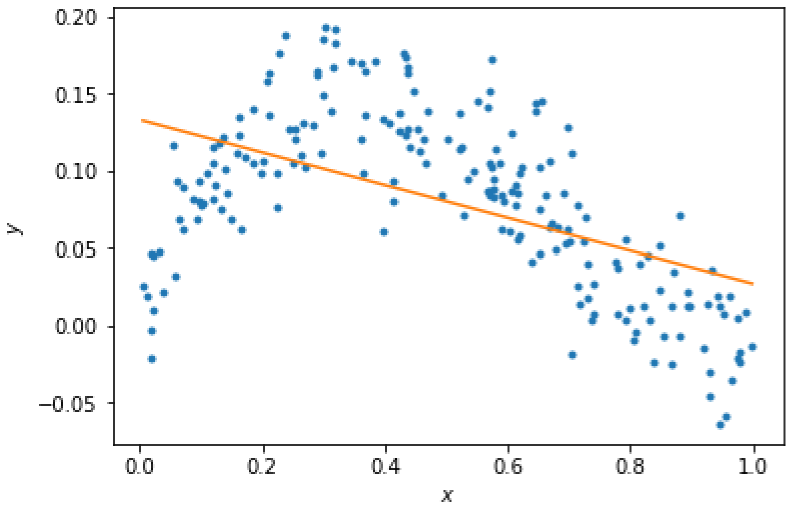
Fitting data to the model (5 points)

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

In [90]:
```
# 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.linalg.inv(xhat.dot(np.transpose(xhat))).dot(xhat).dot(y)

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

In [91]:
```
# 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))
```

Out[91]: [<matplotlib.lines.Line2D at 0x7fe4a27c7c50>]



QUESTIONS

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

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

ANSWERS

(1) The linear model underfits the data.

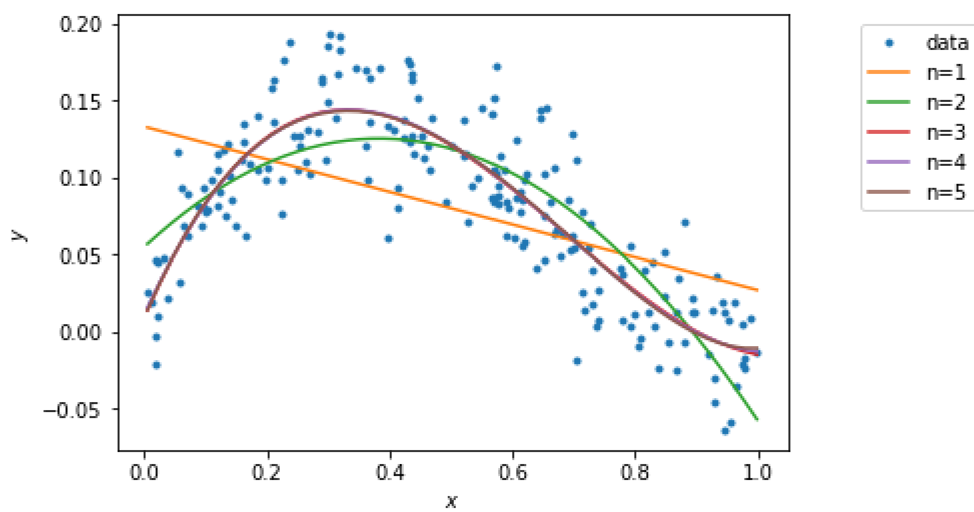
(2) We can make the model a polynomial of degree 3. This way the model would be more flexible and could have a local maximum around $x = 0.3$ and a local minimum at $x = 0.9$.

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.

In [92]:
```
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 po
lynomial 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 coefficient
s of the x^2, x, and 1 respectively.
#   ... etc.
features = [np.ones_like(x)]
for i in range(1, N + 1):
    features = [x**i] + features
    xhats.append(np.vstack(features))

for i in range(N):
    thetas.append(np.linalg.inv(xhats[i].dot(np.transpose(xhats[i]))).
dot(xhats[i]).dot(y))

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

In [93]:
```
# 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)
```



Calculating the training error (10 points)

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

$$L(\theta) = \frac{1}{2}\sum_j (\hat{y}_j - y_j)^2$$

In [94]:
```
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 fi
t of order i+1.

for i in range(N):
    training_errors.append((1/2) * (y.T.dot(y)- 2 * y.T.dot(xhats[i].T
).dot(thetas[i]) + \
    thetas[i].T.dot(xhats[i]).dot(xhats[i].T).dot(thetas[i])))

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

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

```
Training errors are:
 [0.23799610883627054, 0.10924922209268595, 0.08169603801105374, 0.081
653537352969763, 0.081614791955252897]
```

QUESTIONS

(1) Which polynomial model has the best training error?

(2) Why is this expected?

ANSWERS

(1) The polynomial model of degree 5.

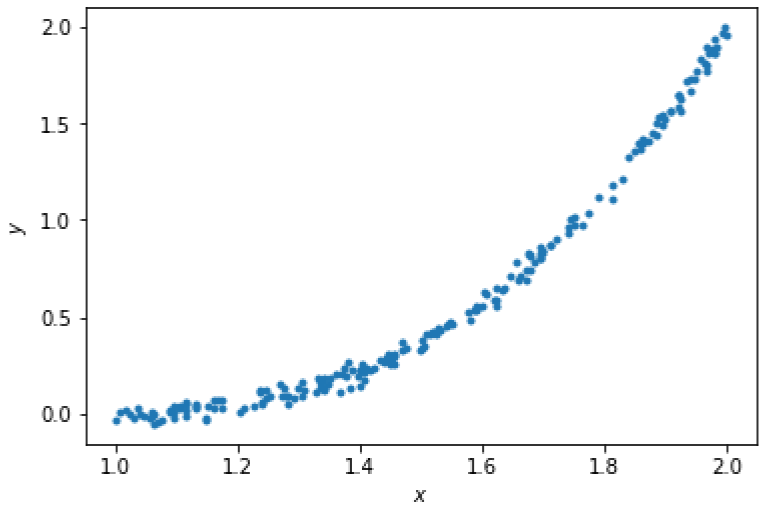
(2) A polynomial of higher degree is more flexible to fit data, because it could potentially have more local optimas (more degrees of freedom). Therefore, we expect a polynomial of degree 5 to be at least as good or even better than any other polynomial of degree 1, 2, 3, or 4.

Generating new samples and testing error (5 points)

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

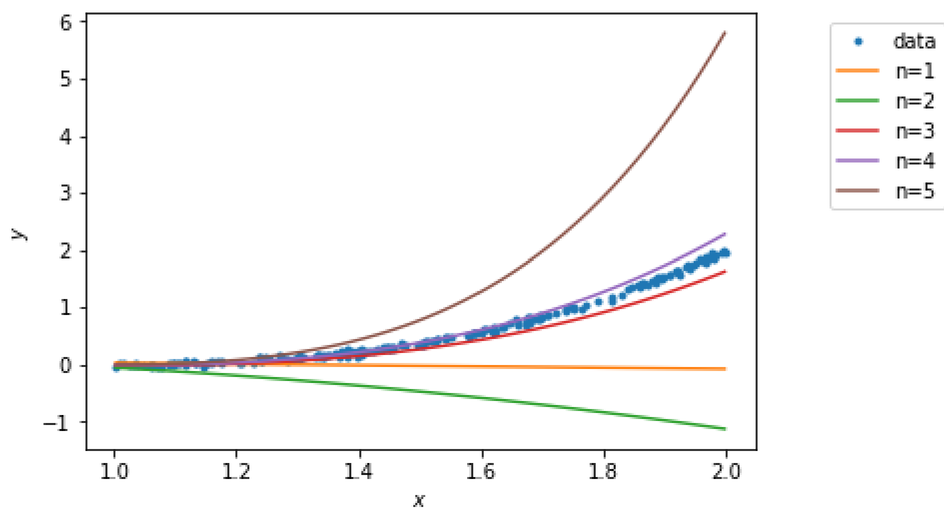
In [95]:
```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_
train,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[95]: Text(0,0.5,'y')



In [96]:
```
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)
```

In [97]:
```
# 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)
```



In [98]:
```
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.

for i in range(N):
    testing_errors.append((1/2) * (y.T.dot(y)- 2 * y.T.dot(xhats[i].T)
.dot(thetas[i]) + \
    thetas[i].T.dot(xhats[i]).dot(xhats[i].T).dot(thetas[i])))

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

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

```
Testing errors are:
 [80.861651845505946, 213.19192445058206, 3.1256971083047063, 1.187076
5193960438, 214.91021837340656]
```

QUESTIONS

(1) Which polynomial model has the best testing error?

(2) Why does the order-5 polynomial model not generalize well?

ANSWERS

(1) The polynomial model of degree 4.

(2) This happens because the order-5 polynomial overfits the data (the model is too complex to generalize given our data distribution).