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

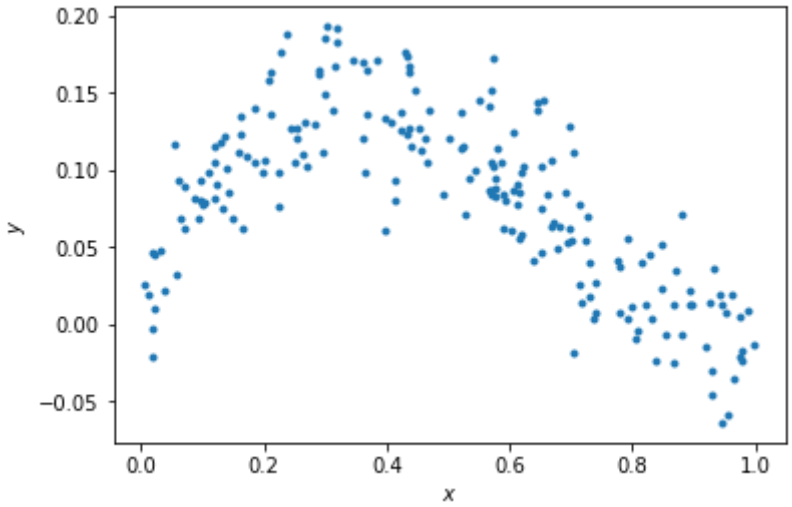
In [97]:
```
import numpy as np
import matplotlib.pyplot as plt

#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 [98]:
```
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_tr
ain,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[98]: 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 follows a uniform distribution between 0 and 1.

(2) ϵ follows a standard normal distribution with mean 0 and standard deviation 0.03.

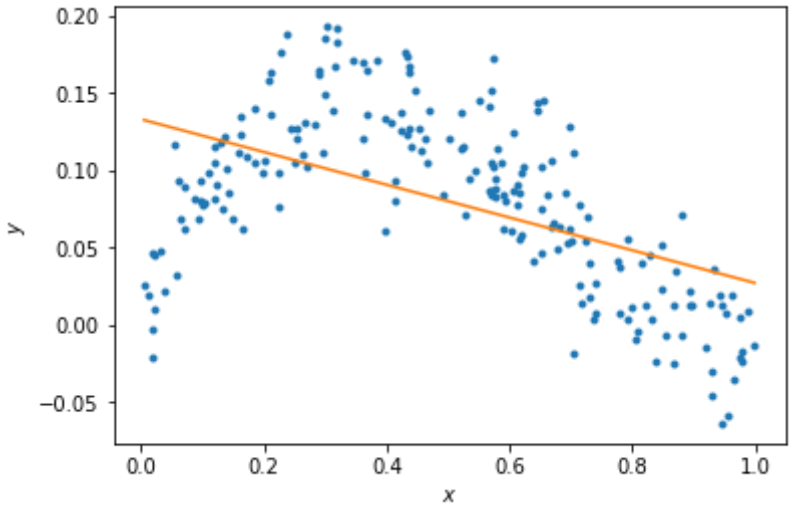
Fitting data to the model (5 points)

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

In [99]:
```
# 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 a
re [a, b]

theta = np.linalg.inv(xhat.dot(np.transpose(xhat))).dot(xhat).dot(y)

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

In [100]:
```
# 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[100]: [<matplotlib.lines.Line2D at 0x10c7cbdd8>]



QUESTIONS

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

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

ANSWERS

(1) It appears to underfit the data because it does not accurately capture the curve in the data.

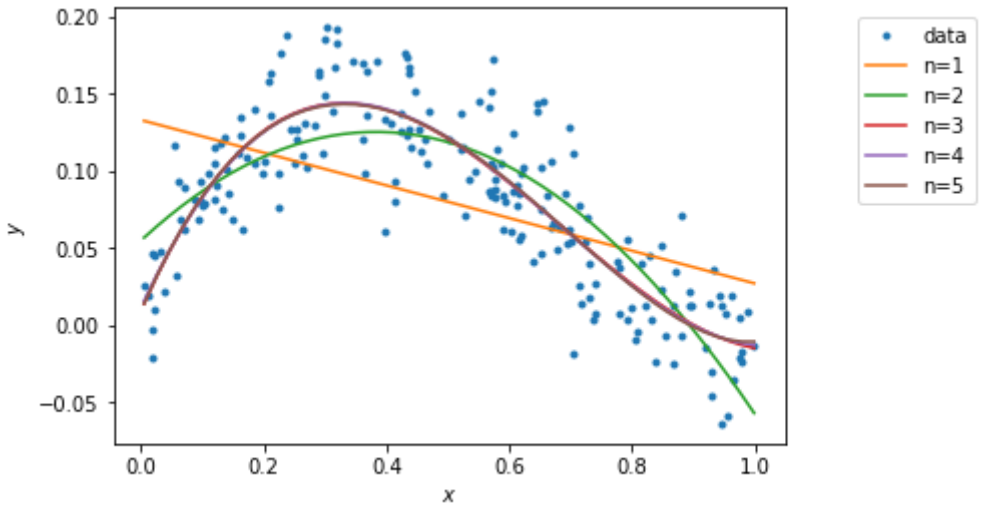
(2) We could increase the complexity of the model to accomodate the curve in the data, perhaps by increasing the degree of the polynomial.

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 [101]:
```
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 poly
nomial 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.

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

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

In [102]:
```
# 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 [103]:
```
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.
for i in range(N):
    yhat = thetas[i].dot(xhats[N-1-i:N+1,])
    training_errors.append((np.sum((yhat - y) ** 2) / 2))

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

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

```
Training errors are:
 [0.23799610883627009, 0.1092492220926853, 0.081696038011053712, 0.0816
53537352969791, 0.08161479195525298]
```

QUESTIONS

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

(2) Why is this expected?

ANSWERS

(1) The polynomial of order 5 had the best training error.

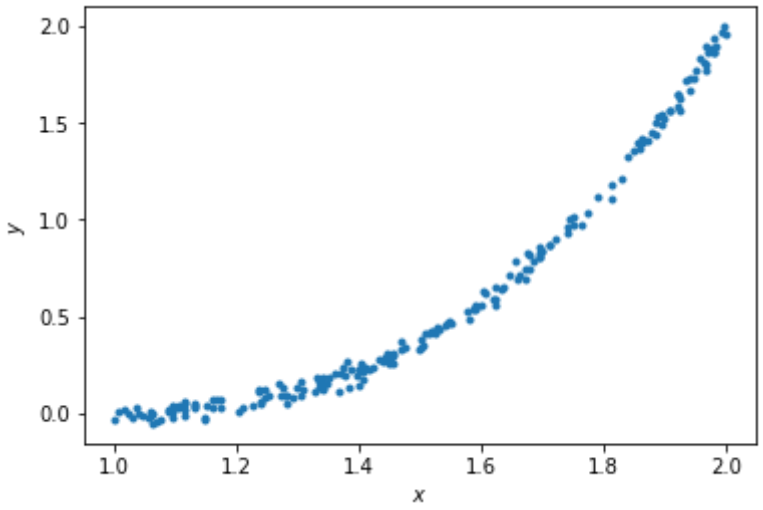
(2) It is expected because it has the most complexity and ability to represent nuances in the training data.

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.

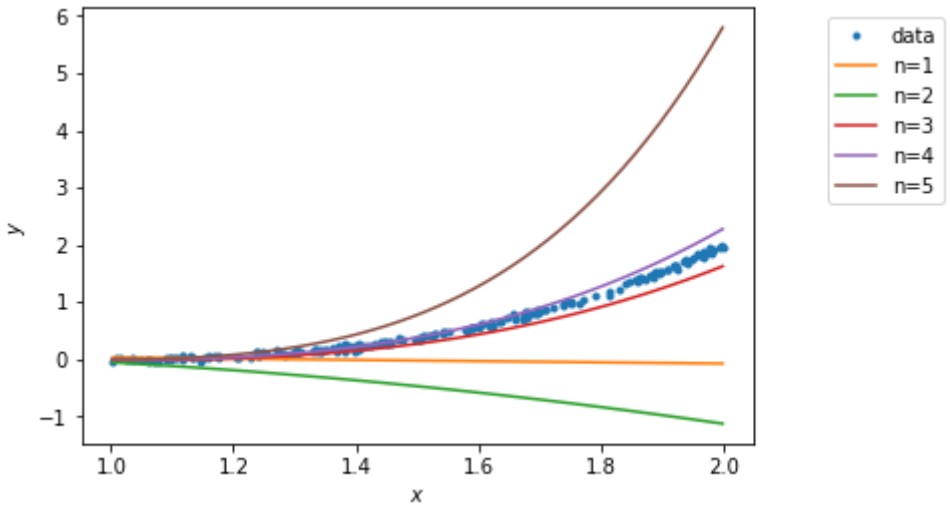
In [104]:
```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_tr
ain,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[104]: Text(0,0.5,'y')



In [105]:
```
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 [106]:
```
# 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 [107]:
```
testing_errors = []

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

for i in range(N):
    yhat = thetas[i].dot(xhats[i])
    testing_errors.append((np.sum((yhat - y) ** 2) / 2))

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

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

```
Testing errors are:
 [80.861651845505889, 213.19192445057996, 3.1256971083286498, 1.1870765
196919209, 214.9102181092214]
```

QUESTIONS

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

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

ANSWERS

(1) The third degree polynomial has the best testing error.

(2) It does not generalize well because it overfits the data, meaning that it is unable to model the underlying distribution. In a sense, it has modeled the training data too faithfully.