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 239AS, Winter Quarter 2018, Prof. J.C. Kao, TAs C. Zhang and T. Xing

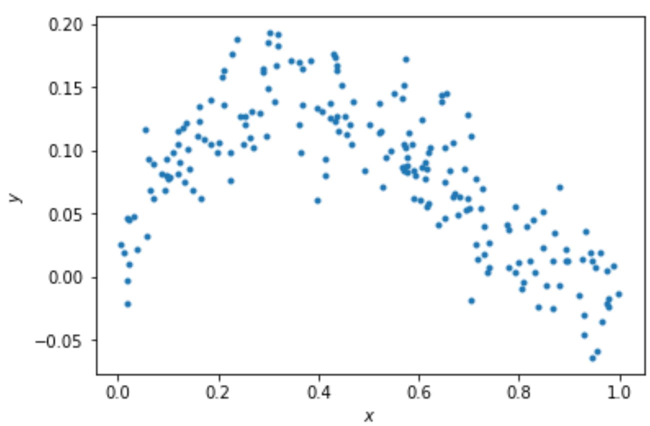
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

#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 [2]:
```
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[2]: `Text(0,0.5,u'$y$')`



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) A uniform distribution between 0 and 1

(2) Normal distribution with mean 0 and std dev 0.03

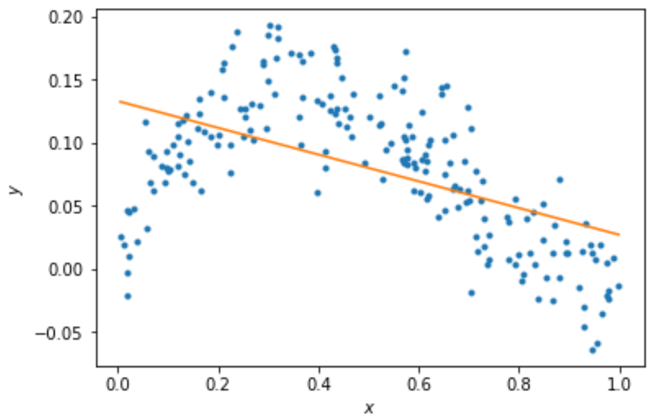
Fitting data to the model (5 points)

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

In [42]:
```
# 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]


#Calculate theta = ((XT*X)^-1)XTY
xhat_t = xhat
xhat_reg = np.transpose(xhat_t)
xhat_mat = xhat_t.dot(xhat_reg)
theta = np.linalg.inv(xhat_mat).dot(xhat_t.dot(y))

# please modify this line

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

In [43]:
```
# 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[43]: [<matplotlib.lines.Line2D at 0x7f440a039190>]



QUESTIONS

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

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

ANSWERS

(1) The model underfits. You can see there is a curve to the data that the linear regression cannot handle

(2) We should increase the complexity of the model by increasing the order of the polynomial

Fitting data to the model (10 points)

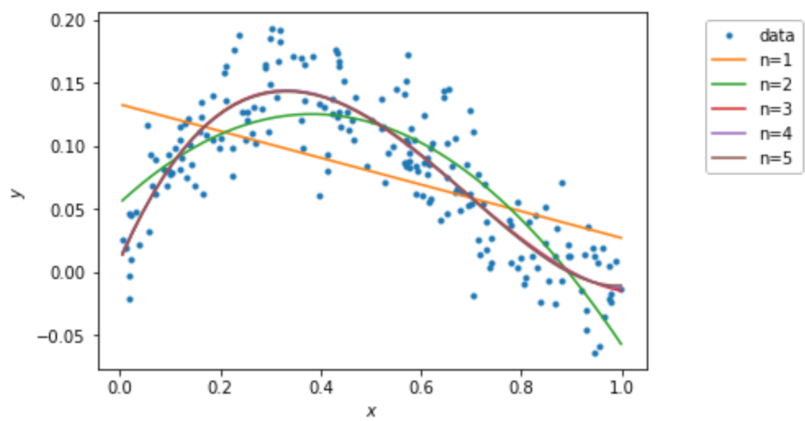
Here, we'll now do regression to polynomial models of orders 1 to 5. Note, the order 1 model is the linear model you prior fit.

In [44]:

```
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 polynom
ial 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)


xhat_p = np.ones_like(x)
xhat_p = np.vstack((x,xhat_p))
xhats.append(xhat_p)


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



pass

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

In [45]:
```
# 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):
    print np.shape(plot_xs[i])
    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)
```

```
(2, 50)
(3, 50)
(4, 50)
(5, 50)
(6, 50)
```



Calculating the training error (10 points)

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

In [47]:
```
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[i])
    error = sum((y-yhat)**2)
    training_errors.append(error)




pass

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

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

```
('Training errors are: \n', [0.47599221767254024, 0.21849844418537057, 0.16
33920760221075, 0.16330707470593955, 0.1632295839105059])
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) Polynomial N=5

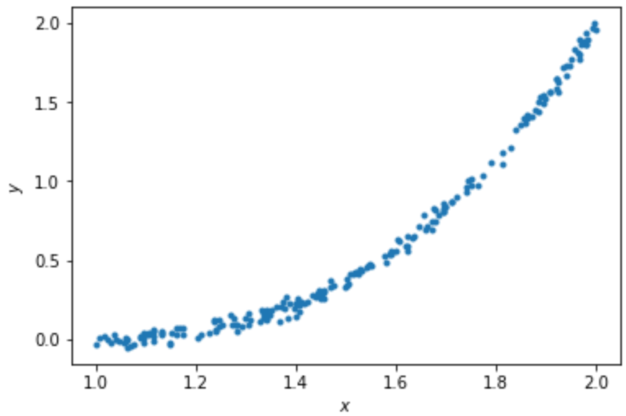
(2) It is overfitting the training data because the complexity allows for it to account for noise

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.

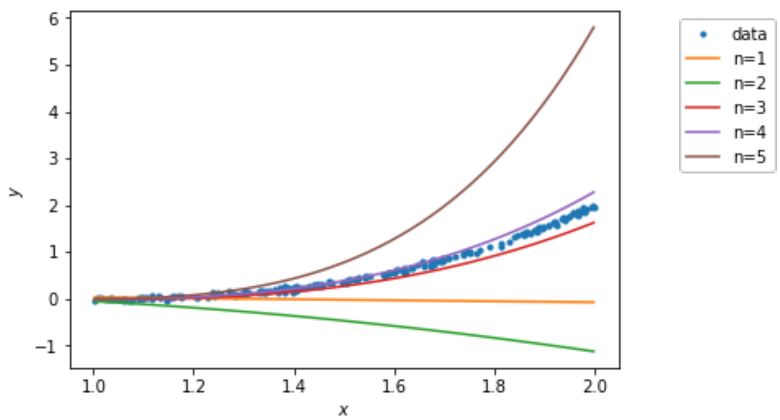
In [48]:
```
x = np.random.uniform(low=1, high=2, size=(num_train,))
y = x - 2*x**2 + x**3 + np.random.normal(loc=0, scale=0.03, size=(num_train
,))
f = plt.figure()
ax = f.gca()
ax.plot(x, y, '.')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

Out[48]: Text(0,0.5,u'y')



In [49]:
```
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 [50]:
```
# 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 [51]:

```
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 or
der i+1.

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


pass

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

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

```
('Testing errors are: \n', [161.7233036910116, 426.38384890115805, 6.251394
2168167524, 2.374153042299244, 429.8204349402559])
```

QUESTIONS

(1) What polynomial has the best testing error?

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

ANSWERS

(1) THe polynomial of order 4 has the best testing error

(2) It overfit the training data and was too specific to the x range of [0,1] that was given in the training batch. This made it perform very well in testing, but lacking when it comes to being general enough to apply to samples it hasn't seen.