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

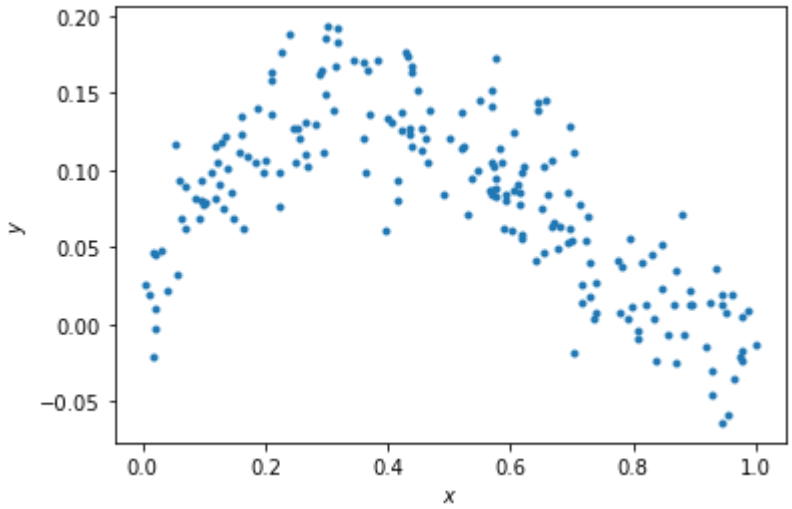
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

#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, '$y$')
```



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) We use `np.random.uniform(low=0, high=1)` to generate x values; hence, the generating distribution of x is a uniform distribution over [0, 1).

(2) We use `np.random.normal(loc=0, scale=0.03)` to generate ϵ values; hence, the distribution of ϵ is a normal (Gaussian) distribution with mean 0 and standard deviation 0.03.

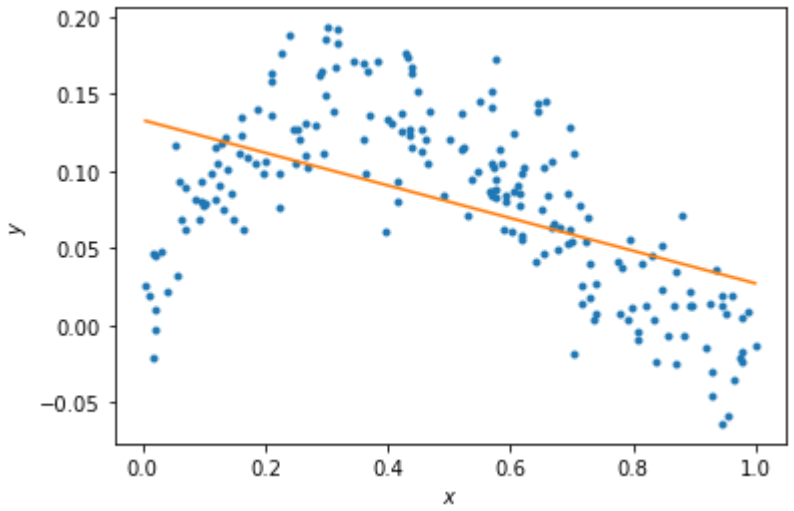
Fitting data to the model (5 points)

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

In [3]:
```
# 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]

#we use least squares formula discussed in lecture 3

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

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

In [4]:
```
# 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))
plt.show()
```



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 the data, since the data is clearly a negative quadratic distribution, not a linear distribution.

(2) We could improve the model by doing polynomial regression instead of linear regression.

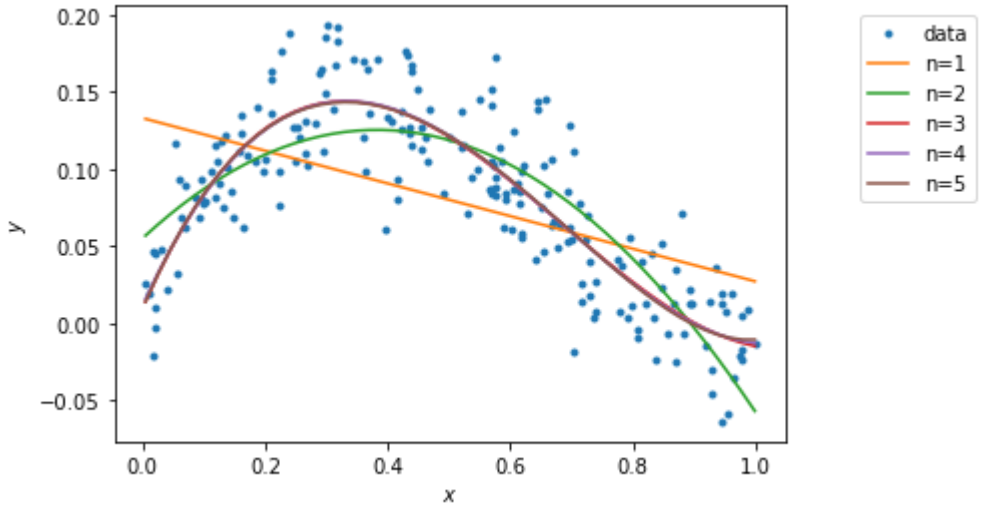
Fitting data to the model (10 points)

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

# ==================== #
# START YOUR CODE HERE #
# ==================== #
xhats.append(xhat)
thetas.append(theta)

for i in range(2, 6):
    xhats.append(np.vstack((x**i, xhats[-1])))
    thetas.append(np.linalg.inv(xhats[-1].dot(xhats[-1].T)).dot(xhats[-1].dot(y)))


# 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.

pass

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

In [6]:
```
# 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.

In [7]:
```
training_errors = []
```

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

for i in range(N):
    predicted_y = thetas[i].dot(xhats[i])
    error = 0.5 * np.sum((predicted_y - y)**2)
    training_errors.append(error)


# 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.
pass

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

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

```
Training errors are:
 [0.2379961088362701, 0.10924922209268531, 0.08169603801105374, 0.08165353735296982, 0.08161479195525298]
```

QUESTIONS

(1) What polynomial has the best training error?

(2) Why is this expected?

ANSWERS

(1) The polynomial with degree 5 has the best/least training data with an error of 0.08161479195525298.

(2) This is expected since a higher degree polynomial will always fit the provided data no worse than a lower degree polynomial, since it can simulate a lower degree polynomial by setting its higher degree coefficients to 0. Looking at the graph, we see that n = 3, 4, and 5 are basically overlapping with one another.

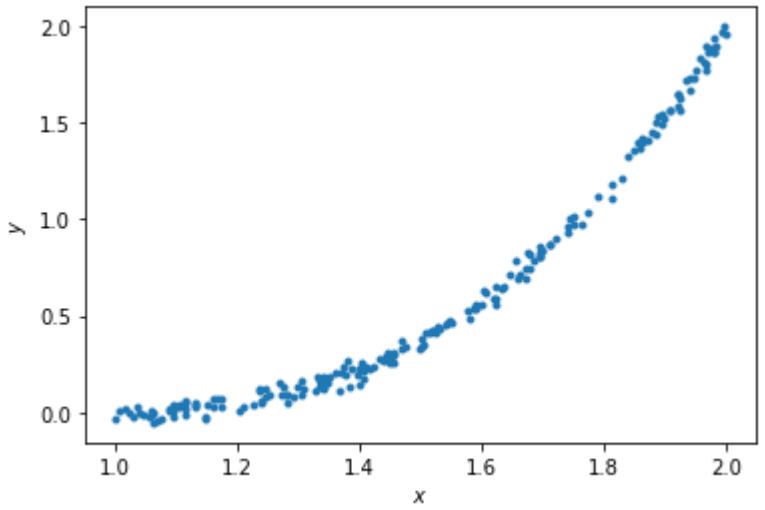
Generating new samples and testing error (5 points)

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

In [8]:
```
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('$y$')
```

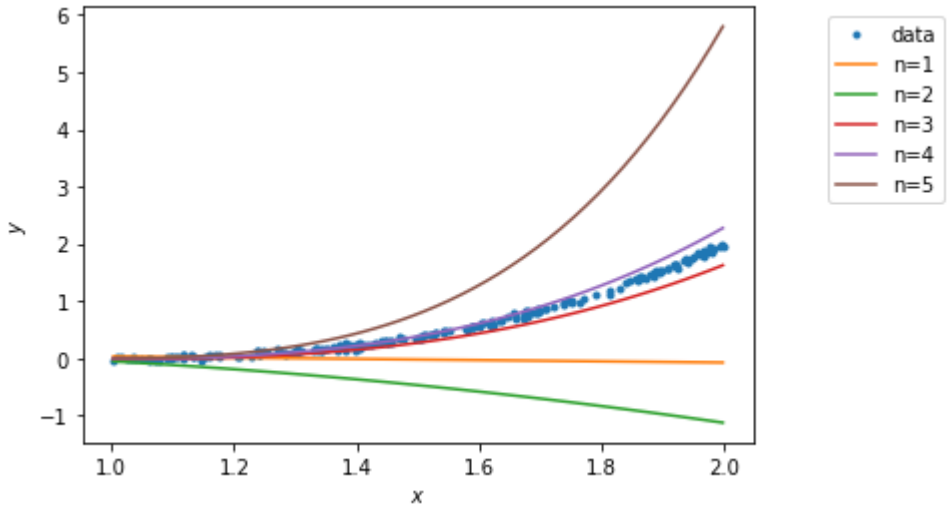
Out[8]: Text(0, 0.5, 'y')



In [9]:
```
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 [10]:
```
# 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 [11]:

```
testing_errors = []

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

for i in range(N):
    predicted_y = thetas[i].dot(xhats[i])
    error = 0.5 * np.sum((predicted_y - y)**2)
    testing_errors.append(error)

# 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.
    pass

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

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

```
Testing errors are:
 [80.86165184550586, 213.19192445057894, 3.1256971082763925, 1.1870765189474703, 214.91021817652626]
```

QUESTIONS

(1) What polynomial has the best testing error?

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

ANSWERS

(1) The polynomial with degree 4 has the best testing error (only 1.1870765189474703), slightly edging the polynomial with degree 3.

(2) Due to overfitting, the polynomial with degree 5 tends to model the noise in the distribution, instead of the underlying trends. Hence, it doesn't generalize well to another data sample of the same distribution, such as the test data set.