DS-210: Programming for Data Science

Lecture 9: Linear regression and its generalizations

Linear regression

Simplest setting

Input: set of points (x_i, y_i) in $\mathbb{R} \times \mathbb{R}$

- What function $f : \mathbb{R} \to \mathbb{R}$ explains the relationship of x_i's with y_i's?
- What linear function $f(x) = ax + b$ describes it best?

Linear regression

Simplest setting

Input: set of points (x_i, y_i) in $\mathbb{R} \times \mathbb{R}$

- What function $f : \mathbb{R} \to \mathbb{R}$ explains the relationship of x_i's with y_i's?
- What linear function $f(x) = ax + b$ describes it best?

Multivariate version

Input: set of points (X_i, y_i) in $\mathbb{R}^d \times \mathbb{R}$

Find linear function
$f(x_1, x_2, \ldots, x_d) = a_1 x_1 + \cdots + a_d x_d + b$ that describes y_i's in terms of X_i's?

Linear regression

Simplest setting

Input: set of points (x_i, y_i) in $\mathbb{R} \times \mathbb{R}$

- What function $f : \mathbb{R} \to \mathbb{R}$ explains the relationship of x_i's with y_i's?
- What linear function $f(x) = ax + b$ describes it best?

Multivariate version

Input: set of points (X_i, y_i) in $\mathbb{R}^d \times \mathbb{R}$

Find linear function $f(x_1, x_2, \ldots, x_d) = a_1 x_1 + \cdots + a_d x_d + b$ that describes y_i's in terms of X_i's?

Why linear regression?

- Have to assume something!
- Models hidden linear relationship + noise

Typical objective: minimize square error

- Points rarely can be described exactly using a linear relationship
- How to decide between several non-ideal options?

Typical objective: minimize square error

- Points rarely can be described exactly using a linear relationship
- How to decide between several non-ideal options?

- Typically want to find f that minimizes total square error:

$$\sum_i (f(x_i) - y_i)^2$$

Typical objective: minimize square error

- Points rarely can be described exactly using a linear relationship
- How to decide between several non-ideal options?

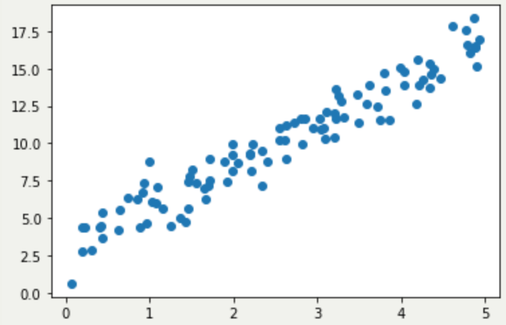
- Typically want to find f that minimizes total square error:

$$\sum_i (f(x_i) - y_i)^2$$

- You should see a linear–algebra based solution for this in your math class

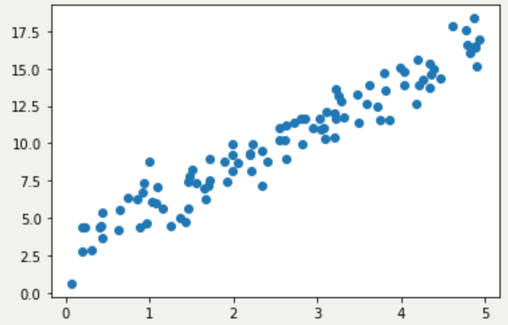
Examples for $d = 1$

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

# generate data
SAMPLES = 100
A,B = 0.0,5.0
x = np.random.uniform(low=A,high=B,size=SAMPLES)
y = x * math.e + math.pi \
    + np.random.normal(size=SAMPLES)
plt.plot(x,y,"o");
```



Examples for $d = 1$

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

# generate data
SAMPLES = 100
A,B = 0.0,5.0
x = np.random.uniform(low=A,high=B,size=SAMPLES)
y = x * math.e + math.pi \
    + np.random.normal(size=SAMPLES)
plt.plot(x,y,"o");
```

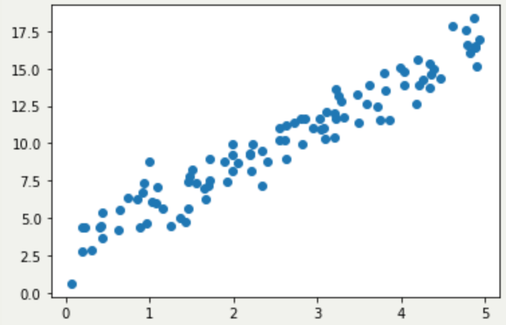


Using `numpy.polyfit`

Finds the best **polynomial** of degree bounded by the third parameter

Examples for $d = 1$

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

# generate data
SAMPLES = 100
A,B = 0.0,5.0
x = np.random.uniform(low=A,high=B,size=SAMPLES)
y = x * math.e + math.pi \
    + np.random.normal(size=SAMPLES)
plt.plot(x,y,"o");
```



Using `numpy.polyfit`

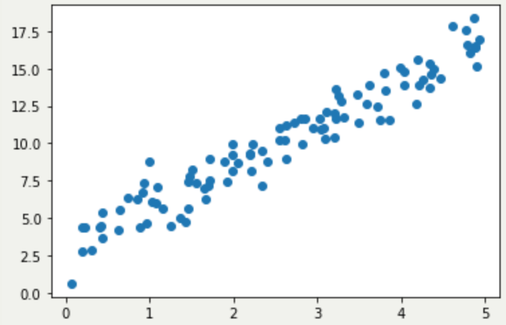
Finds the best **polynomial** of degree bounded by the third parameter

In [2]:
```
c = np.polyfit(x,y,1)
c
```

Out[2]:
```
array([2.76116246, 2.98740734])
```

Examples for $d = 1$

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

# generate data
SAMPLES = 100
A,B = 0.0,5.0
x = np.random.uniform(low=A,high=B,size=SAMPLES)
y = x * math.e + math.pi \
    + np.random.normal(size=SAMPLES)
plt.plot(x,y,"o");
```



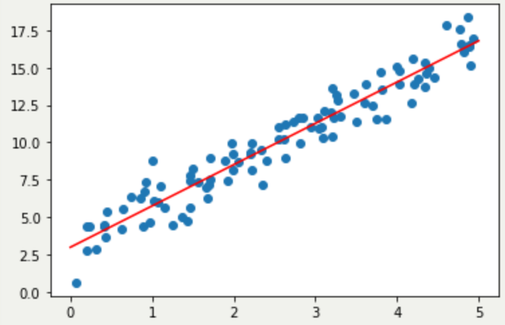
Using `numpy.polyfit`

Finds the best **polynomial** of degree bounded by the third parameter

In [2]:
```
c = np.polyfit(x,y,1)
c
```

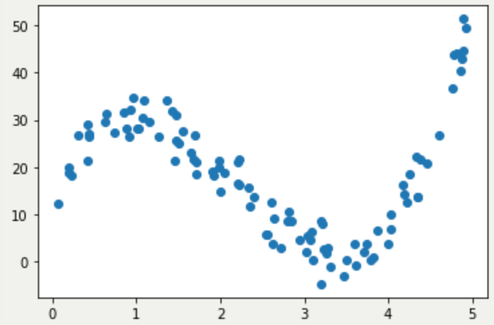
Out[2]:
```
array([2.76116246, 2.98740734])
```

In [3]:
```
reg_x = np.linspace(A,B,num=100)
reg_y = reg_x * c[0] + c[1]
plt.plot(x,y,"o",reg_x,reg_y,"r");
```



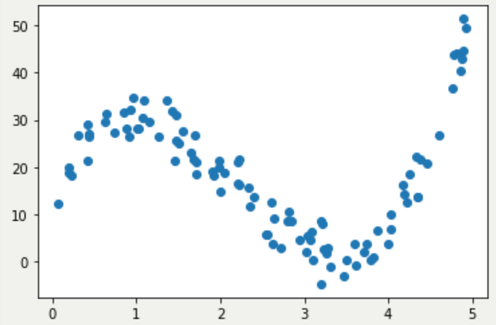
`numpy.polyfit`: fitting higher degree polynomials

In [4]:
```
y2 = x**3 * 4 - x**2 * 26 + x * 38 + \
    15 + 3.0 * np.random.normal(size=SAMPLES)
plt.plot(x,y2,"o");
```

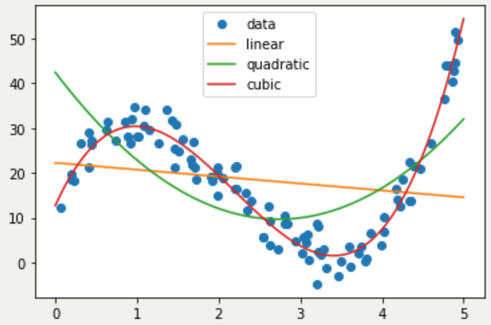


`numpy.polyfit`: fitting higher degree polynomials

In [4]:
```
y2 = x**3 * 4 - x**2 * 26 + x * 38 + \
    15 + 3.0 * np.random.normal(size=SAMPLES)
plt.plot(x,y2,"o");
```



In [5]:
```
def eval_poly(c,x):
    y = np.zeros_like(x)
    for coeff in c:
        y = y * x + coeff
    return y

C = [np.polyfit(x,y2,i) for i in range(1,4)]
REG_Y = [eval_poly(c,reg_x) for c in C]
plt.plot(x,y2,"o")
for reg_y in REG_Y:
    plt.plot(reg_x,reg_y)
plt.legend(['data','linear','quadratic','cubic'],
           loc="upper center");
```



Examples for $d = 1$: using `scipy.stats.linregress`

In [6]:
```
from scipy.stats import linregress
result = linregress(x,y)
result
```

Out[6]:
```
LinregressResult(slope=2.761162456495789, intercept=2.9874073369170056, rvalue=0.9631897127859851, pvalue=1.0204975643243524e-57, stderr=0.07784537662642257, intercept_stderr=0.2261580518953048)
```

Examples for $d = 1$: using `scipy.stats.linregress`

```
In [6]: from scipy.stats import linregress
        result = linregress(x,y)
        result

Out[6]: LinregressResult(slope=2.761162456495789, intercept=2.9874073369170056, rvalue=0.9631897127859851, pvalue=1.0204975643243524e-5
        7, stderr=0.07784537662642257, intercept_stderr=0.2261580518953048)
```

- `slope` $= a$
- `intercept` $= b$ in

$$f(x) = ax + b$$

Examples for $d = 1$: using `scipy.stats.linregress`

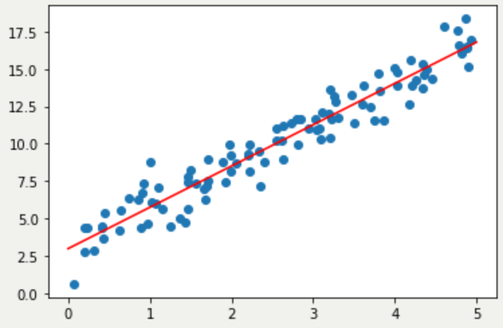
In [6]:
```
from scipy.stats import linregress
result = linregress(x,y)
result
```

Out[6]:
```
LinregressResult(slope=2.761162456495789, intercept=2.9874073369170056, rvalue=0.9631897127859851, pvalue=1.0204975643243524e-57, stderr=0.07784537662642257, intercept_stderr=0.2261580518953048)
```

- `slope` $= a$
- `intercept` $= b$ in

$$f(x) = ax + b$$

In [7]:
```
reg_x = np.array([A,B])
reg_y = reg_x * result.slope + result.intercept
plt.plot(x,y,"o",reg_x,reg_y,"-r");
```



Coefficient of determination (or R^2)

- How good is my function f?

Coefficient of determination (or R^2)

- How good is my function f?

- **Input:** points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- **Idea:** Compare variance of y_i's to the deviation of y_i's from $f(x_i)$'s

Coefficient of determination (or R^2)

- How good is my function f?

- **Input:** points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- **Idea:** Compare variance of y_i's to the deviation of y_i's from $f(x_i)$'s

- **Formally:**

$$1 - \frac{\sum_i (y_i - f(x_i))^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\frac{1}{n}\sum_i (y_i - f(x_i))^2}{\mathrm{Var}(y_i)}$$

where $\bar{y} = \frac{1}{n}\sum_i y_i$

- **Range:** $(-\infty, 1]$ (should be in $[0, 1]$ for linear regression)

Coefficient of determination (or R^2)

- How good is my function f?

- **Input:** points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- **Idea:** Compare variance of y_i's to the deviation of y_i's from $f(x_i)$'s

- **Formally:**

$$1 - \frac{\sum_i (y_i - f(x_i))^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\frac{1}{n}\sum_i (y_i - f(x_i))^2}{\mathrm{Var}(y_i)}$$

where $\bar{y} = \frac{1}{n}\sum_i y_i$

- **Range:** $(-\infty, 1]$ (should be in $[0, 1]$ for linear regression)

Intuition: What fraction of y_i's variance is explained by f?

Compute for our data!

$$1 - \frac{\sum_i (y_i - f(x_i))^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\frac{1}{n}\sum_i (y_i - f(x_i))^2}{\mathrm{Var}(y_i)}$$

In [8]:
```
variance = y.var()
mse = ((x*result.slope + result.intercept - y)**2).mean()
variance,mse
```

Out[8]: (15.981172256908307, 1.1548886372104468)

In [9]:
```
1 - mse/variance
```

Out[9]: 0.9277344228167483

Multiple dimensions with scikit-learn

In [10]:
```
# d = 2
SAMPLES = 100
x = np.random.normal(size=(SAMPLES,2))
x[:5]
```

Out[10]:
```
array([[-0.47028336,  0.97588483],
       [-1.50639013,  0.21537361],
       [ 0.55192326, -2.44915707],
       [-0.18020037,  0.75093991],
       [-0.52431573, -0.85205588]])
```

Multiple dimensions with scikit-learn

In [10]:
```
# d = 2
SAMPLES = 100
x = np.random.normal(size=(SAMPLES,2))
x[:5]
```

Out[10]:
```
array([[-0.47028336,  0.97588483],
       [-1.50639013,  0.21537361],
       [ 0.55192326, -2.44915707],
       [-0.18020037,  0.75093991],
       [-0.52431573, -0.85205588]])
```

In [11]:
```
y = np.matmul(x,[2.3,1.4]) + 3.1 + \
    np.random.normal(size=SAMPLES)
y[:5]
```

Out[11]:
```
array([ 3.54780877, -1.11677711,  0.71313861,  2.8525289
1,  0.85431898])
```

Multiple dimensions with scikit-learn

In [10]:
```
# d = 2
SAMPLES = 100
x = np.random.normal(size=(SAMPLES,2))
x[:5]
```

Out[10]:
```
array([[-0.47028336,  0.97588483],
       [-1.50639013,  0.21537361],
       [ 0.55192326, -2.44915707],
       [-0.18020037,  0.75093991],
       [-0.52431573, -0.85205588]])
```

In [11]:
```
y = np.matmul(x,[2.3,1.4]) + 3.1 + \
    np.random.normal(size=SAMPLES)
y[:5]
```

Out[11]:
```
array([ 3.54780877, -1.11677711,  0.71313861,  2.8525289
1,  0.85431898])
```

In [12]:
```
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(x,y)
reg.coef_,reg.intercept_
```

Out[12]:
```
(array([2.26233205, 1.40991034]), 3.0885671373997106)
```

Categorical attributes to numbers: How?

Categorical attributes to numbers: How?

- Does `cat` / `dog` / `cow` / `fish` ⇒ `0` / `1` / `2` / `3` work well for linear regression?

Categorical attributes to numbers: How?

- Does `cat`/`dog`/`cow`/`fish` ⇒ `0`/`1`/`2`/`3` work well for linear regression?

- Do this instead:
  - Introduce a Boolean `0`/`1` variable for each category
  - `is_a_cat`/`is_a_dog`/`is_a_cow`/`is_a_fish`?
  - You can skip one of them

Generalization with `scipy.optimize.leasts_squares`

- You write your own function that takes a parameter and returns a vector of errors
- `least_squares` will optimize the parameter for you
- second parameter is the starting point for optimization

In [13]:
```
from scipy.optimize import least_squares

def error(c):
    return c - 3

least_squares(error,0)
```

Out[13]:
```
 active_mask: array([0.])
        cost: 0.0
         fun: array([0.])
        grad: array([0.])
         jac: array([[1.]])
     message: '`gtol` termination condition is satisfie
d.'
        nfev: 3
        njev: 3
  optimality: 0.0
      status: 1
     success: True
           x: array([3.])
```

This example: Finding c that minimizes

$$(c-3)^2$$

Solution = attribute `x` in the output of `least_squares`

`least_squares`: Two dimensional parameter

In [14]:
```
def error(c):
    return (c[0] - math.pi)**2 + (c[1] - math.e)**2

least_squares(error,[0,0])
```

Out[14]:
```
 active_mask: array([0., 0.])
        cost: 5.4760369667382635e-18
         fun: array([3.30939178e-09])
        grad: array([-2.87782824e-13, -2.49005810e-13])
         jac: array([[-8.69594304e-05, -7.52421674e-0
5]])
     message: '`gtol` termination condition is satisfie
d.'
        nfev: 17
        njev: 9
  optimality: 2.8778282405172454e-13
      status: 1
     success: True
           x: array([3.14154915, 2.71824419])
```

This example: Finding (c_0, c_1) that minimize

$$\left((c_0 - \pi)^2 + (c_1 - e)^2\right)^2$$

Solution = attribute `x` in the output of `least_squares`

`least_squares`: Linear regression

So far the output was a scalar, but here it's a vector: the error for each data point.

In [15]:
```
SAMPLES = 100
x = np.random.normal(size=(SAMPLES,2))
y = np.matmul(x,[2.3,1.4]) + 3.1 + \
    np.random.normal(size=SAMPLES)

def error(c):
    return np.matmul(x,c[:2]) + c[2] - y

least_squares(error,[0.0,0.0,0.0]).x
```

Out[15]:
```
array([2.37521909, 1.42037847, 3.09188993])
```

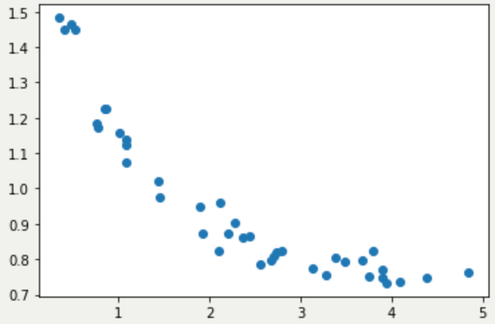
This example: Minimizing

$$\sum_i (c_0 \cdot x[i][0] + c_1 \cdot x[i][1] + c_2 - y[i])^2$$

Attribute `x` of `least_squares`'s output is an approximately best (c_0, c_1, c_2)

`least_squares`: Find best $f(x) = 2^{\alpha x} + \beta$

In [16]:
```
SAMPLES = 40
A,B = 0.0,5.0
x = np.random.uniform(low=A,high=B,size=SAMPLES)
y = 2**(x*-1.1) + 0.7\
    + np.random.normal(scale = 0.05, size=SAMPLES)
plt.plot(x,y,"o");
```



In [17]:
```
def error(c):
    return 2**(x*c[0]) + c[1] - y

solution = least_squares(error,[0,0]).x
print(solution)

reg_x = np.linspace(A,B,100)
reg_y = 2**(reg_x*solution[0]) + solution[1]
plt.plot(x,y,"o",reg_x,reg_y,"r-");
```

```
[-1.27902646  0.73458113]
```