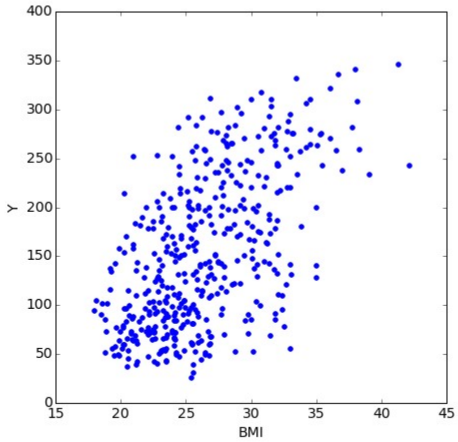
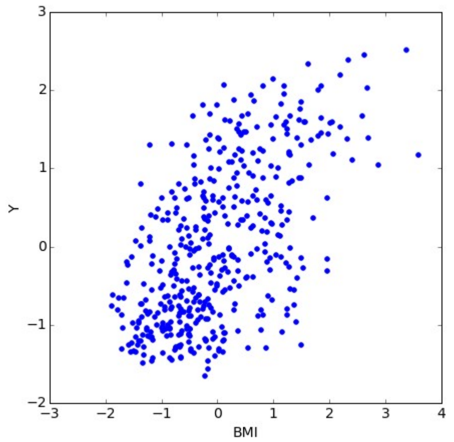
regression



- Can BMI explain Y?
- Can BMI predict Y?
- How does Y vary with BMI?

Regression is a general term for **modeling the relationship** between a **label** (a.k.a. dependent variable or target) and one or more **features** (a.k.a. the attributes, the independent or explanatory variable(s)).

linear regression



We need to make

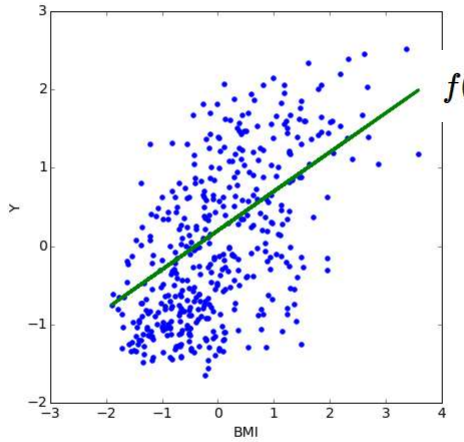
assumptions *linear relationship*

about the

model *linear model*

that generated the data.

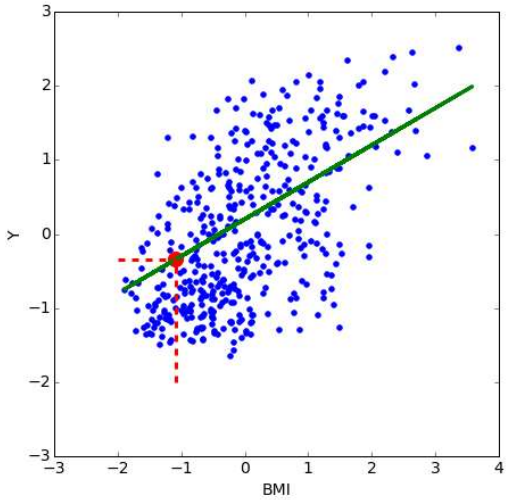
linear regression



$$f(x) = 0.5 \cdot x + 0.2 = ax + b$$

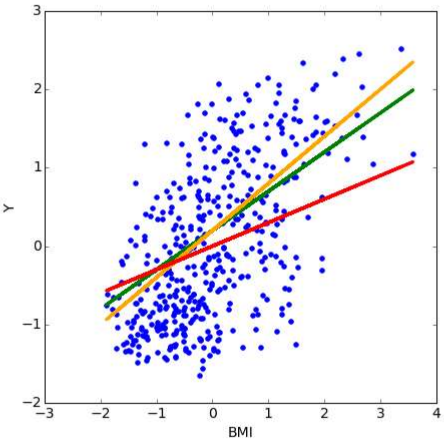
- *a* and *b* are model parameters
- *a* is the slope to the line (direction)
- *b* is the intercept or bias (position)

linear regression: prediction



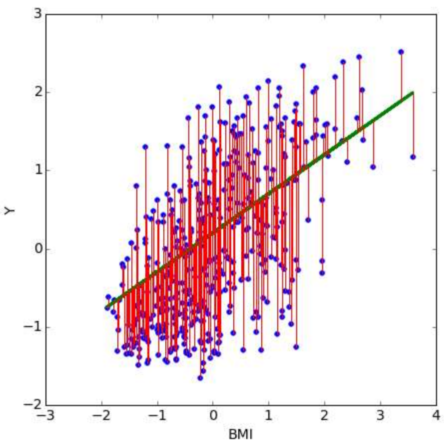
- BMI(70kg, 1.8 meter) = 21.6
- scaled data!
- scaled BMI = -1.08
- predicted value = f(-1.08)
- predicted Y = 125.9

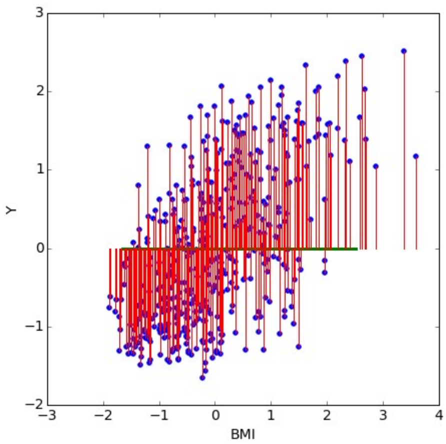
linear regression: fitting



- What values for *a* and *b* fit the data best?
- How can we evaluate them?

linear regression: fitting





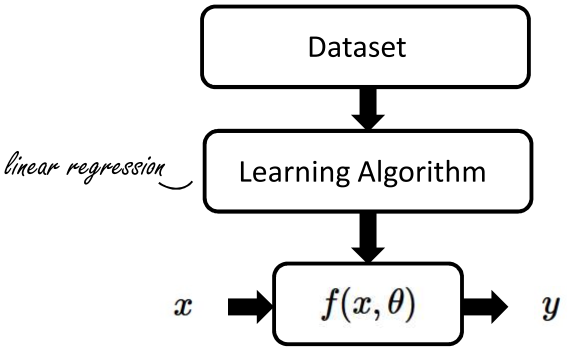
$$R^2 = 1 - \frac{SS_{error}}{SS_{total}}$$

$$R^2 = 0.296$$

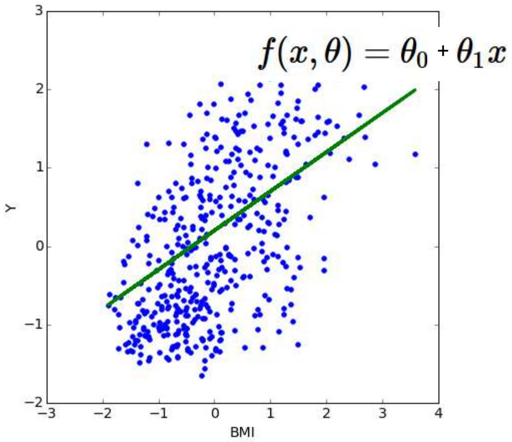
$$SS_{error} = \sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}))^2$$

$$SS_{total} = \sum_{i=1}^{n} (y^{(i)} - \bar{y})^2$$

linear regression



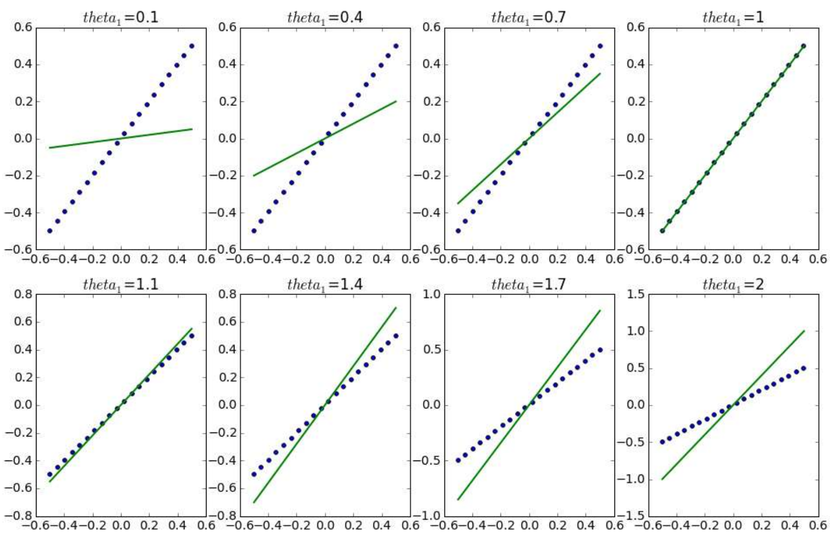
$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m = \theta' x$$



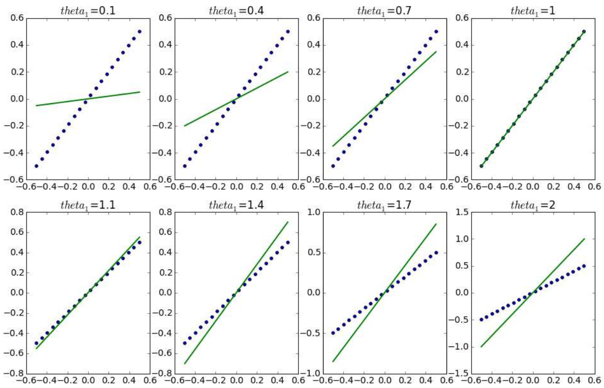
linear regression: cost

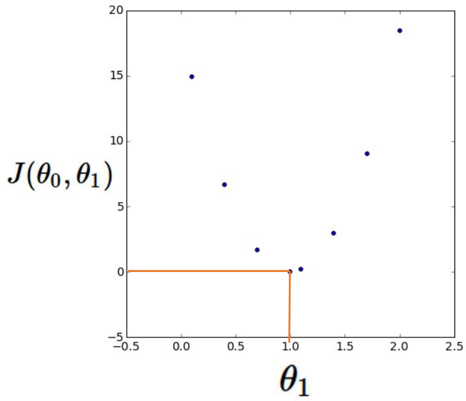
$$f(x) = x$$

$$(\theta_1 = 1, \theta_0 = 0)$$



linear regression: cost





$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)})^2$$

linear regression

Fit a linear model

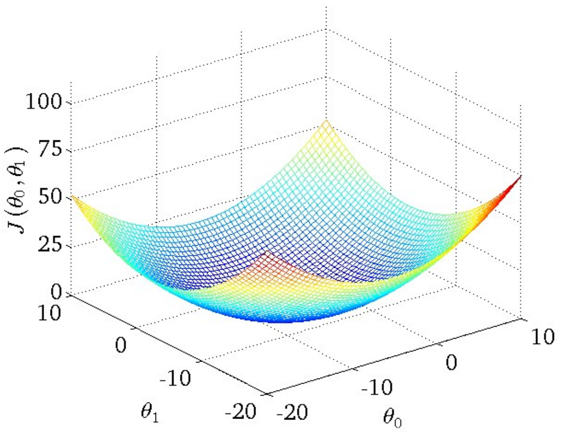
$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m = \theta' x$$

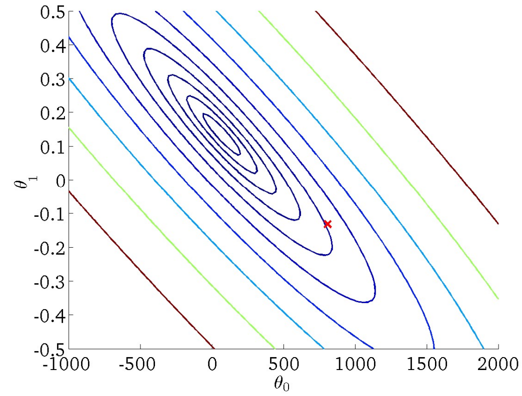
to the data set such that the cost function

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)})^2$$

is minimal.

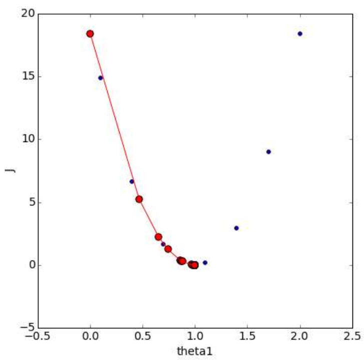
linear regression

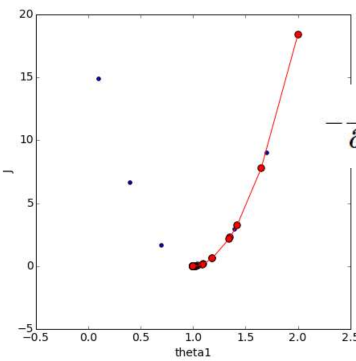




gradient descent

1. start with some random initialization of θ, i.e. a randomly chosen model
2. increment or decrement the value(s) of θ slightly such that $J(\theta)$ is reduced
3. repeat step 2. until we observe convergence





$$-\frac{\partial}{\partial \theta_j} J(\theta)$$

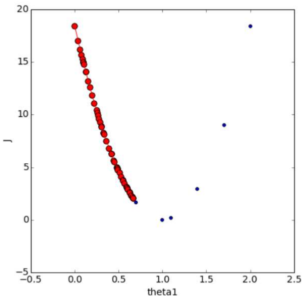
gradient descent

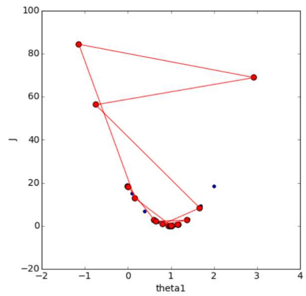
$$\theta_0 := \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_2^{(i)}$$

$$\dots$$





linear regression

model: $f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$

cost function: $J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)})^2$

Goal: $\underset{J(\theta)}{\textbf{minimize}} \quad J(\theta)$

Learning: 1. start with some θ
2. change θ to reduce $J(\theta)$
3. repeat 2. until convergence

$$\theta_0 := \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{n} \sum_{i=1}^{n} (f(x^{(i)}, \theta) - y^{(i)}) x_2^{(i)}$$

...

```python
def compute_R_squared(X,y,a,b):
    E = ((y - (a*X+b))**2).sum()
    V = ((y - y.mean())**2).sum()
    return 1.0 - (E/V)

print "R-squared = %f" % compute_R_squared(dataset['BMI'],dataset['Y'],a,b)
```

```
R-squared = 0.296450
```

```python
from sklearn import metrics

print "R-squared = %f" % metrics.r2_score(dataset['Y'],a*dataset['BMI']+b)
```

```
R-squared = 0.296450
```

```python
# Function performs linear regression using gradient descent.
# Parameters:
# X = feature vectors
# y = target
# alpha = gradient descent learning rate
# iterations = number of iterations in gradient descent
def linear_regression(X,y,alpha,iterations):
    # intialize theta1 with random value
    theta1 = 3
    #theta1 = -3
    for i in range(iterations):
        # select random feature vector
        next = np.random.randint(len(X))
        # predict target
        predict = np.dot(X[next],theta1)
        # compute prediction error
        error = predict - y[next]
        # update theta1 such that cost function decreases
        theta1 = theta1 - alpha*error*X[next]
    return theta1
```

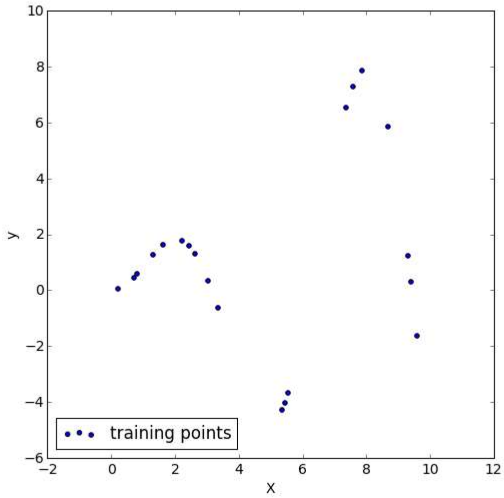
```
from sklearn import linear_model

#eta0 is the learning rate used for gradient descent
model = linear_model.SGDRegressor(eta0=0.001)
model.fit(dataset_scaled,target)

print "R-squared = %f" % metrics.r2_score(target,model.predict(dataset_scaled))
```

```
R-squared = 0.419080
```

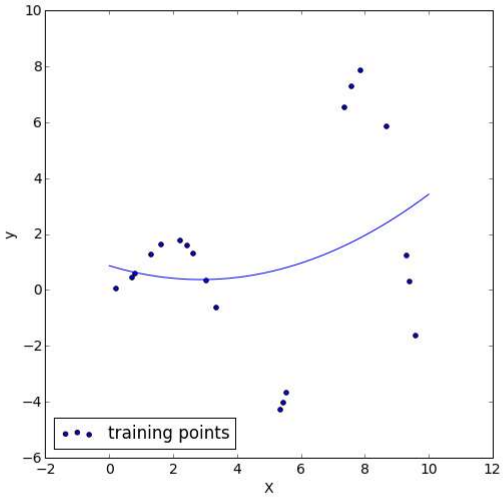
non-linear regression



- Does *y* vary linearly with *X*?
- Can we fit a non-linear model? Yes
- Or we could add polynomial transformations of the features.

$$f(x, \theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$$

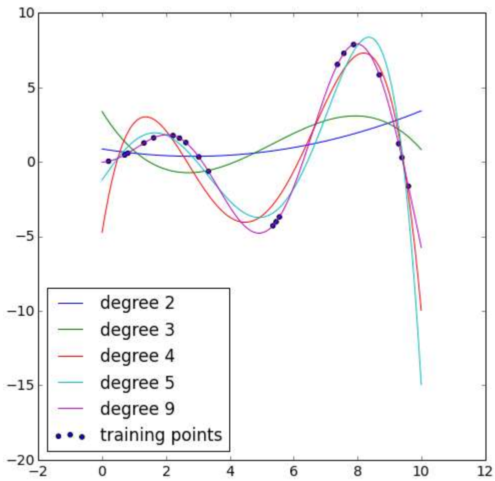
non-linear regression



- Does *y* vary linearly with *X*?
- Can we fit a non-linear model? Yes
- Or we could add polynomial transformations of the features.

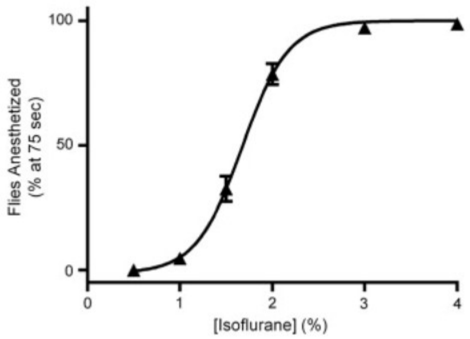
$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$$

non-linear regression



$$f(x,\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \theta_4 x_1^4 + \theta_5 x_1^5 + \theta_6 x_1^6 + \theta_7 x_1^7 + \theta_8 x_1^8 + \theta_9 x_1^9$$

non-linear regression



- dose-response relationship
- sigmoid function (a.k.a. a logistic function)

$$f(x) = \frac{1}{1 + e^{-\theta_1(x-\theta_0)}}$$

- theta1 is the slope at the steepest part of the curve
- theta0 is the dosage at which 50% of the subjects are expected to show the desired response

```
print dataset.head()
dataset['x1^2'] = dataset['x1']**2
print "New dataset:"
print dataset.head()
sns.lmplot(x="x1", y="x1^2", data=dataset,
           fit_reg=False, size=5, scatter_kws={"s": 80})
plt.show()
```

```
         x1         y
0  0.202020  0.040535
1  0.707071  0.459320
2  0.808081  0.584212
3  1.313131  1.269782
4  1.616162  1.614499
New dataset:
         x1         y      x1^2
0  0.202020  0.040535  0.040812
1  0.707071  0.459320  0.499949
2  0.808081  0.584212  0.652995
3  1.313131  1.269782  1.724314
4  1.616162  1.614499  2.611978
```

```
sns.lmplot(x="x1", y="y", data=dataset,
           fit_reg=False, size=5, scatter_kws={"s": 80})

X = dataset.copy()
y = X.pop('y')

model = LinearRegression(fit_intercept=True)
x_plot = np.linspace(0, 10, 100)

for degree in [3, 4, 5, 6, 7, 8, 9]:
    X['x1^'+str(degree)] = X['x1']**degree

    model.fit(X, y)

    pred = model.intercept_
    for i in range(degree):
        pred += model.coef_[i]*((x_plot)**(i+1))
    plt.plot(x_plot,pred,label="%d" % degree)
plt.legend(loc='lower left')
plt.show()
```