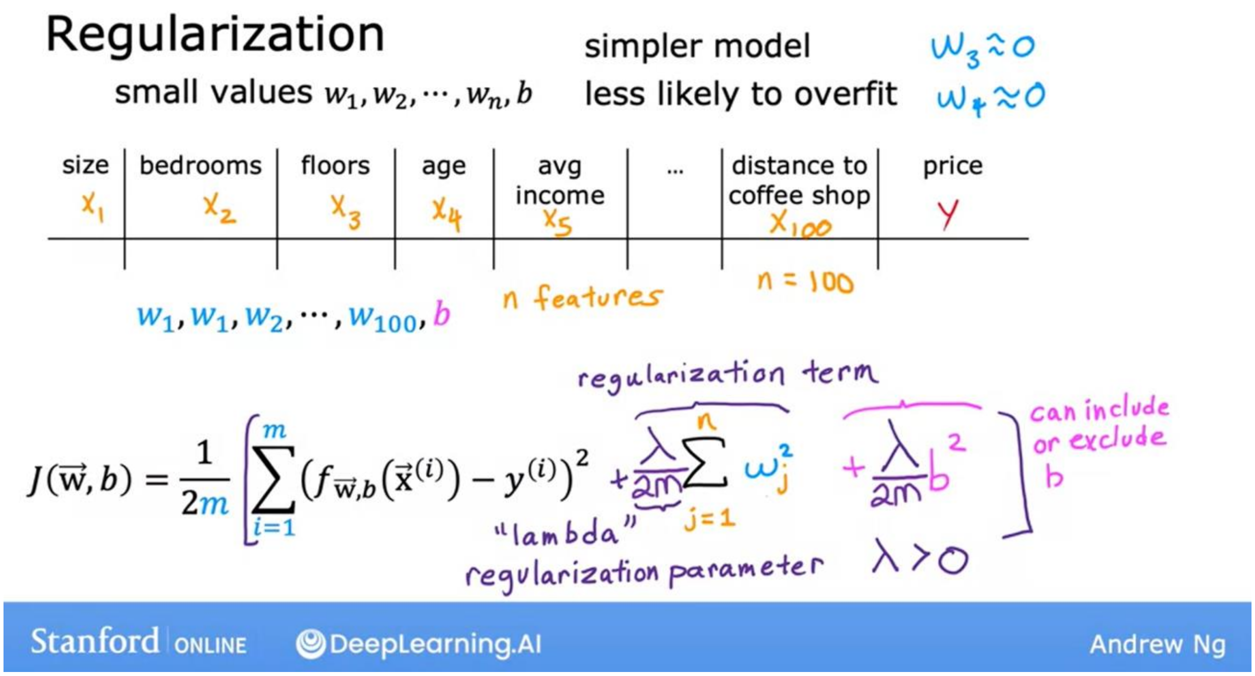


Including this term encourages gradient descent to minimize the size of the parameters. Note, in this example, the parameter *bb* is not regularized. This is standard practice.

if lambda is 0 this model will over fit If lambda is enormous like 10 to the power of 10. This model will under fit.

And so what you want is some value of lambda that is in between that more appropriately balances these first and second terms of trading off, minimizing the mean squared error and keeping the parameters small. And when the value of lambda is not too small and not too large, but just right, then hopefully you end up able to fit a 4th order polynomial, keeping all of these features, but with a function that looks like this.

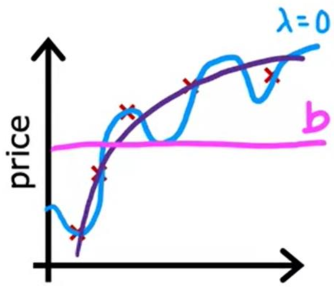
Regularization

$$\min_{\vec{w},b} J(\vec{w},b) = \min_{\vec{w},b}\left[\underbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}_{\text{mean squared error}} + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}_{\text{regularization term}}\right]$$

fit data

Keep w_j small

λ balances both goals



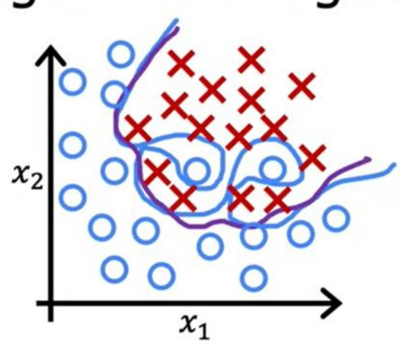
choose $\lambda = 10^{10}$

$$f_{\vec{w},b}(\vec{x}) = \underbrace{\cancel{w_1}}_{\approx 0}x + \underbrace{\cancel{w_2}}_{\approx 0}x^2 + \underbrace{\cancel{w_3}}_{\approx 0}x^3 + \underbrace{\cancel{w_4}}_{\approx 0}x^4 + b$$

$f(x) = b$

Choose λ

Regularized logistic regression



$$z = w_1x_1 + w_2x_2 + w_3x_1^2x_2 + w_4x_1^2x_2^2 + w_5x_1^2x_2^3 + \cdots + b$$

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-z}}$$

Cost function

$$J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

$\min_{\vec{w},b} J(\vec{w},b) \rightarrow w_j \downarrow$

Regularized logistic regression

$$J(\vec{\mathrm{w}},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

$\min_{\vec{w},b}$

Gradient descent

Looks same as for linear regression!

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}},b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left[\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)x_j^{(i)}\right] + \frac{\lambda}{m} w_j$$

$j = 1 \ldots n$

logistic regression

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}},b) \qquad = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}(\vec{\mathrm{x}}^{(i)}) - y^{(i)}\right)$$

}

don't have to regularize

Gradient function for regularized logistic regression

```
[9]: def compute_gradient_logistic_reg(X, y, w, b, lambda_):
    """
    Computes the gradient for linear regression

    Args:
      X (ndarray (m,n): Data, m examples with n features
      y (ndarray (m,)): target values
      w (ndarray (n,)): model parameters
      b (scalar)      : model parameter
      lambda_ (scalar): Controls amount of regularization
    Returns
      dj_dw (ndarray Shape (n,)): The gradient of the cost w.r.t. the parameters w.
      dj_db (scalar)            : The gradient of the cost w.r.t. the parameter b.
    """
    m,n = X.shape
    dj_dw = np.zeros((n,))                           #(n,)
    dj_db = 0.0                                      #scalar
```

```python
for i in range(m):
    f_wb_i = sigmoid(np.dot(X[i],w) + b)          #(n,)(n,)=scalar
    err_i  = f_wb_i  - y[i]                       #scalar
    for j in range(n):
        dj_dw[j] = dj_dw[j] + err_i * X[i,j]      #scalar
    dj_db = dj_db + err_i
dj_dw = dj_dw/m                                   #(n,)
dj_db = dj_db/m                                   #scalar

for j in range(n):
    dj_dw[j] = dj_dw[j] + (lambda_/m) * w[j]

return dj_db, dj_dw
```