Machine Learning & Artificial Intelligence for Data Scientists: Regression (Part 3)

Ke Yuan
https://kyuanlab.org/
School of Computing Science

What happens to the parameters when a complex model is overfitting?

In [234]:
```
for i in range(1, 10, 2):
    poly_order = i
    X_train = make_polynomial(x, poly_order)
    poly_reg = LinearRegression().fit(X_train, t)
    param = poly_reg.coef_[0]
    param[0] = poly_reg.intercept_[0]
    print("order: ", i, ":", param )
```

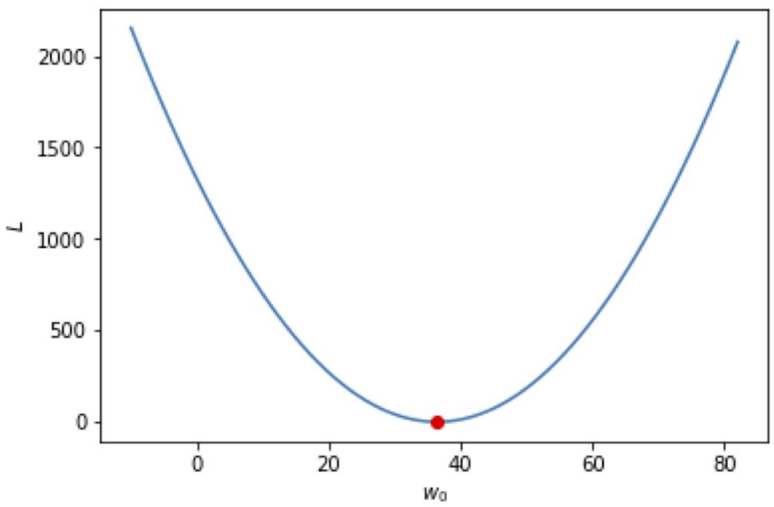
```
order:  1 : [11.14109659 -0.53323543]
order:  3 : [11.52829666 -1.95586746  1.02257133 -0.19981031]
order:  5 : [11.76096702 -4.53086023  6.99555283 -5.37006803  1.88766668 -0.24
62754 ]
order:  7 : [ 11.88431396  -8.2542977   26.51293334 -44.03421185  38.5909018
 -18.249853     4.40637879  -0.42604153]
order:  9 : [  11.98032237  -15.41186085   86.9871579  -240.58612568  363.1578
9942
 -322.96642614  174.03310952  -55.86414073    9.82843863   -0.72955703]
```

We don't want the parameters to be too big in absolute value

```
In [39]: L = np.zeros(num_candidates) # preallocating a vector for L e.g. np.zeros
         for j in range(num_candidates): # For loop to evaluate L at every w0_candidates wi
         th w1 = -0.013330885711.
             L[j] =  np.mean( (t-w0_candidates[j]-(-0.013330885711)*x)**2 )

         plt.plot(w0_candidates, L)
         plt.plot(36.4164559025,0.05, 'ro')
         plt.xlabel('$w_0$')
         plt.ylabel('$L$')

Out[39]: Text(0, 0.5, '$L$')
```



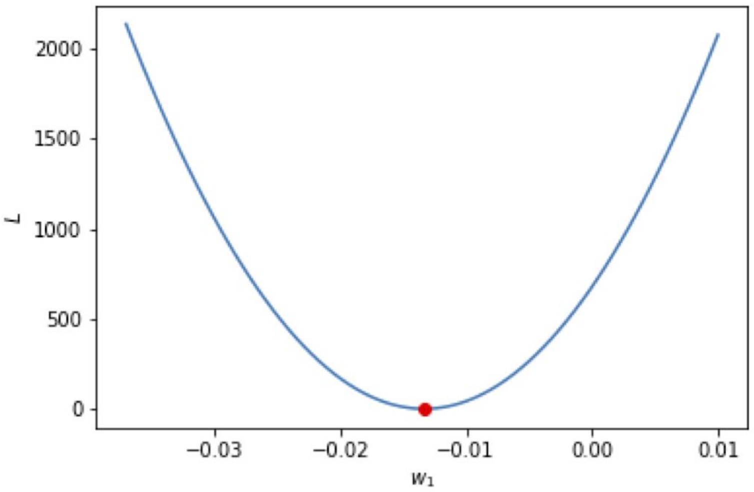
Let's look at the loss function in 1D

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N} (\mathbf{t} - \mathbf{X}\mathbf{w})^T (\mathbf{t} - \mathbf{X}\mathbf{w})$$

In [40]:
```
L = np.zeros(num_candidates) # preallocating a vector for L e.g. np.zeros
for j in range(num_candidates): # For loop to evaluate L at every w1_candidates wi
th w0 = 36.4164559025
    L[j] =  np.mean( (t-36.4164559025-w1_candidates[j]*x)**2 ) # You can use the
 "np.mean" function

plt.plot(w1_candidates, L) # plot the resulting
plt.plot(-0.013330885711,0.05, 'ro')
plt.xlabel('$w_1$')
plt.ylabel('$L$')
```

Out[40]:
```
Text(0, 0.5, '$L$')
```



Let's look at the loss function in 1D

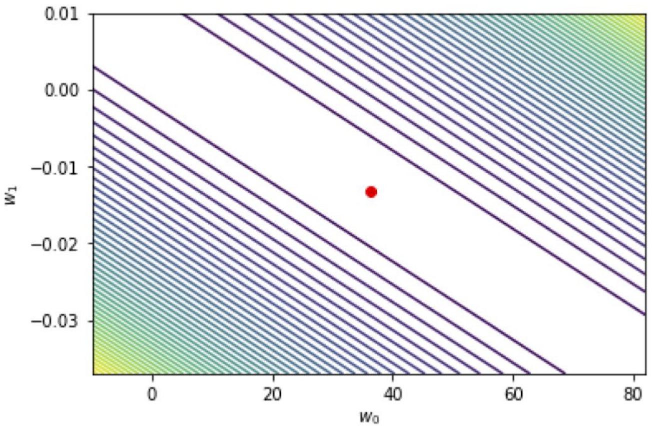
$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N} (\mathbf{t} - \mathbf{X}\mathbf{w})^T (\mathbf{t} - \mathbf{X}\mathbf{w})$$

In [41]:
```python
L = np.zeros( shape = (num_candidates,num_candidates) ) # Prelocate the loss. We a
re going to have num_candidates times num_candidates of them

# Two nested for loops
for i in range(num_candidates):
    for j in range(num_candidates):
        L[i,j] = np.mean( (t-w0_candidates[i]-w1_candidates[j]*x)**2 )


X, Y = np.meshgrid(w0_candidates, w1_candidates) # Make the x and y coordinates fo
r contour plot
plt.contour(X, Y, L, 50)
plt.plot(36.4164559025, -0.013330885711,'ro')
plt.xlabel('$w_0$')
plt.ylabel('$w_1$')
```

Out[41]:
```
Text(0, 0.5, '$w_1$')
```

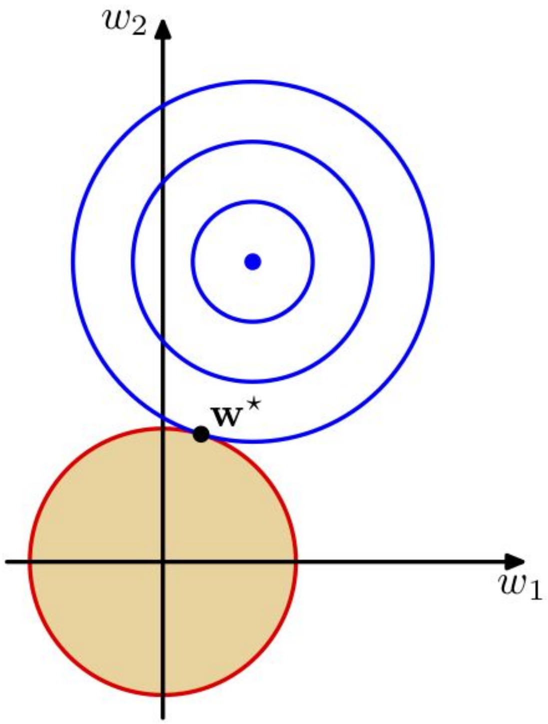


And in 2D

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N} (\mathbf{t} - \mathbf{X}\mathbf{w})^T (\mathbf{t} - \mathbf{X}\mathbf{w})$$

Regularised linear regression: Ridge regression

$$\hat{\mathbf{w}}_{ridge} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w}) + \alpha \mathbf{w}^T \mathbf{w}$$

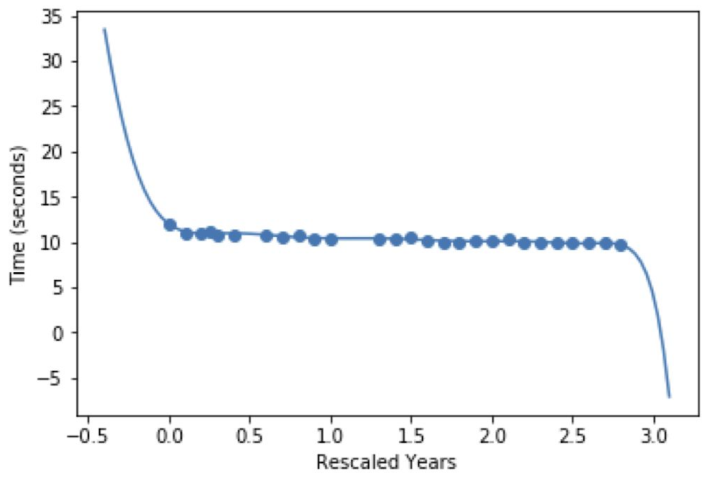


In [87]:
```python
poly_order = 10
X_train = make_polynomial(x, poly_order)
X_test = make_polynomial(x_test, poly_order)

reg = LinearRegression()
reg.fit(X_train, t)

plt.plot(x_test, reg.predict(X_test))
plt.scatter(x,t) # draw a scatter plot
plt.xlabel('Rescaled Years') # always label x&y-axis
plt.ylabel('Time (seconds)') # always label x&y-axis
```

Out[87]:
```
Text(0, 0.5, 'Time (seconds)')
```



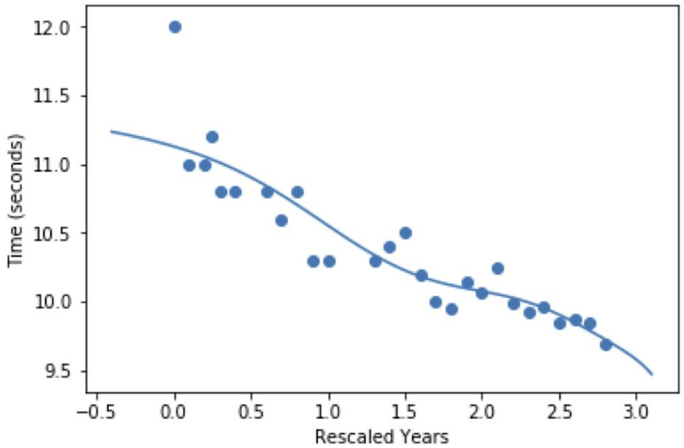
Fit a polynomial regression model of order 10

In [88]:
```python
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ridge = Ridge()
parameters = {'alpha': np.linspace(1, 10, 20)}
ridge_model = GridSearchCV(ridge, parameters, scoring = 'neg_mean_squared_error',
cv=5)
ridge_model.fit(X_train, t)

plt.plot(x_test, ridge_model.predict(X_test))
plt.scatter(x,t) # draw a scatter plot
plt.xlabel('Rescaled Years') # always label x&y-axis
plt.ylabel('Time (seconds)') # always label x&y-axis
```

Out[88]:
```
Text(0, 0.5, 'Time (seconds)')
```



Fit a ridge regression model with \alpha determined by 5-fold CV

Compare parameters between linear and ridge regression

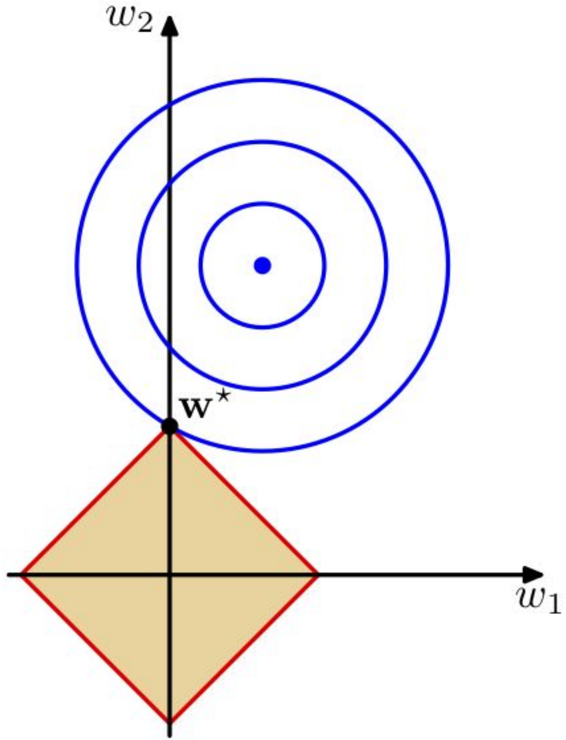
In [89]:

```
param_lr = reg.coef_[0]
param_ridge = ridge_model.best_estimator_.coef_[0]
param_lr[0] = reg.intercept_[0]
param_ridge[0] = ridge_model.best_estimator_.intercept_[0]
print(np.hstack((param_lr[:,None], param_ridge[:,None])) )
```

```
[[ 1.19648635e+01  1.11285803e+01]
 [-1.29443055e+01 -3.29359603e-01]
 [ 5.79522897e+01 -1.94725736e-01]
 [-1.09582035e+02 -9.64898104e-02]
 [ 6.23248849e+01 -1.87327081e-02]
 [ 7.48519704e+01  3.32402164e-02]
 [-1.46955431e+02  4.50182751e-02]
 [ 1.04735797e+02  9.53751777e-03]
 [-3.88035781e+01 -3.60588365e-02]
 [ 7.43343695e+00  1.40369595e-02]
 [-5.82870289e-01 -1.62830483e-03]]
```

Regularised linear regression: Lasso

$$\hat{\mathbf{w}}_{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w}) + \alpha \sum_d |w_d|$$



In [90]:

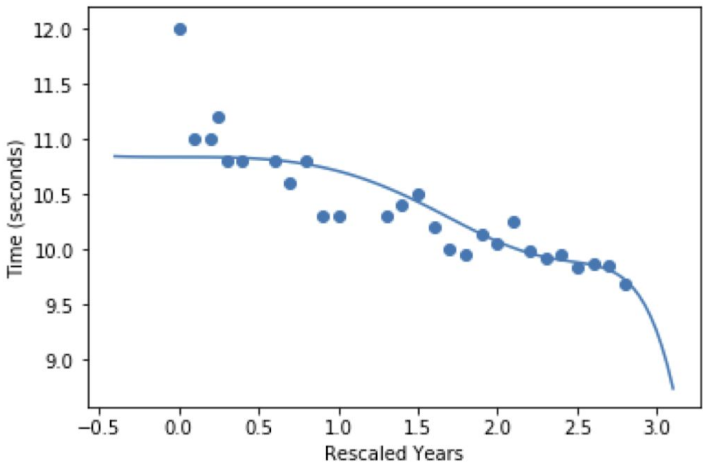
```
from sklearn.linear_model import Lasso

lasso = Lasso()
parameters = {'alpha': np.linspace(1e-1, 1, 20)}
lasso_model = GridSearchCV(lasso, parameters, scoring = 'neg_mean_squared_error',
cv=5)
lasso_model.fit(X_train, t)

plt.plot(x_test, lasso_model.predict(X_test))
plt.scatter(x,t) # draw a scatter plot
plt.xlabel('Rescaled Years') # always label x&y-axis
plt.ylabel('Time (seconds)') # always label x&y-axis
```

Out[90]:

```
Text(0, 0.5, 'Time (seconds)')
```



Fit a Lasso model with \alpha determined by 5-fold CV

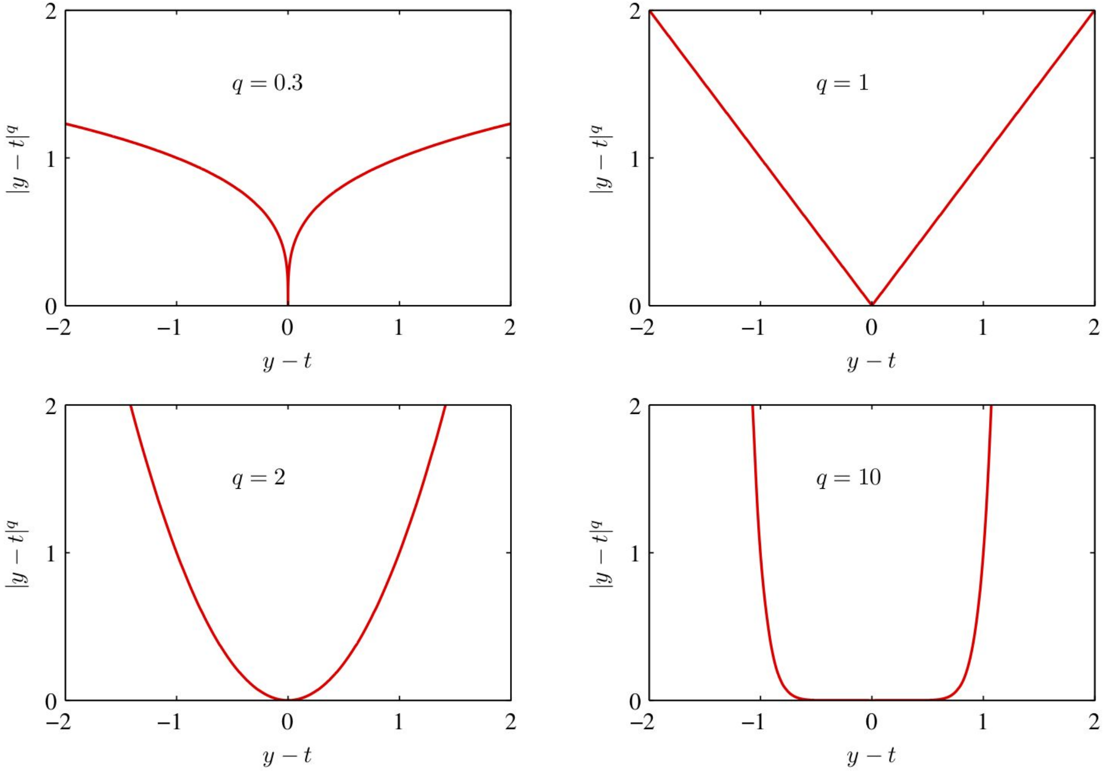
Compare parameters between linear regression, ridge regression, Lasso

```
In [91]: param_lasso = lasso_model.best_estimator_.coef_
         param_lasso[0] = lasso_model.best_estimator_.intercept_[0]
         print(np.hstack((param_lr[:,None], param_ridge[:, None], param_lasso[:,None])) )
```

```
[[ 1.19648635e+01  1.11285803e+01  1.08381535e+01]
 [-1.29443055e+01 -3.29359603e-01 -0.00000000e+00]
 [ 5.79522897e+01 -1.94725736e-01 -0.00000000e+00]
 [-1.09582035e+02 -9.64898104e-02 -1.16126582e-01]
 [ 6.23248849e+01 -1.87327081e-02 -1.59001968e-02]
 [ 7.48519704e+01  3.32402164e-02 -0.00000000e+00]
 [-1.46955431e+02  4.50182751e-02  1.38119952e-03]
 [ 1.04735797e+02  9.53751777e-03  3.22128802e-03]
 [-3.88035781e+01 -3.60588365e-02  1.61616847e-04]
 [ 7.43343695e+00  1.40369595e-02 -8.65262203e-05]
 [-5.82870289e-01 -1.62830483e-03 -7.74413350e-05]]
```

What about a different loss?

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n}^{N} |t_n - \mathbf{w}^T \mathbf{x}_n|^q$$



Should we care about what is x?