Regression 2

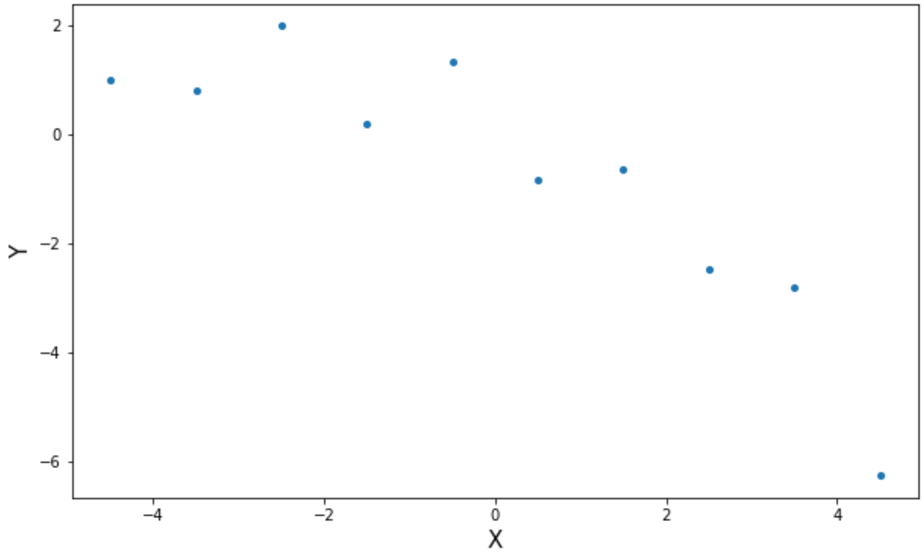
Industrial AI Lab.

Advanced Linear Regression

- Overfitting
- Linear Basis Function Models
- Regularization (Ridge and Lasso)
- Evaluation

Overfitting

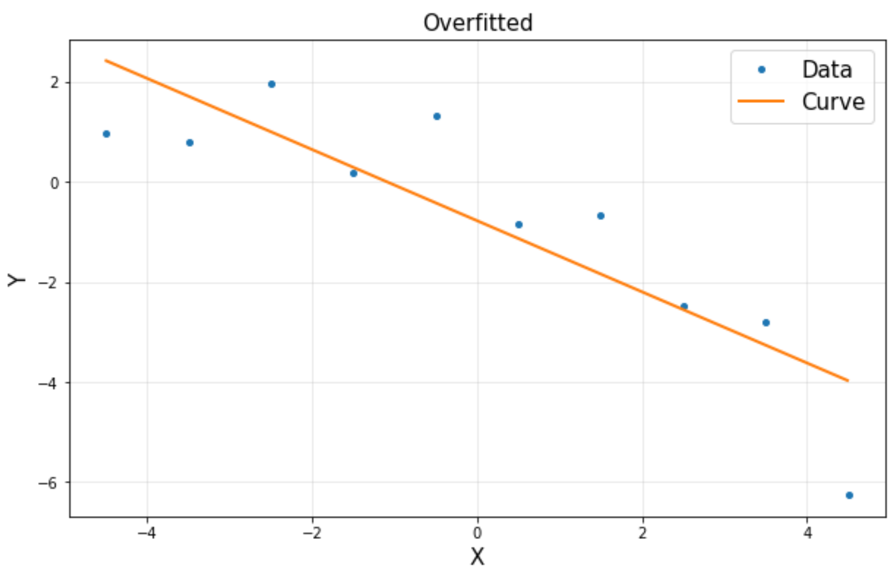
```
# 10 data points
n = 10
x = np.linspace(-4.5, 4.5, 10).reshape(-1, 1)
y = np.array([0.9819, 0.7973, 1.9737, 0.1838, 1.3180, -0.8361, -0.6591, -2.4701,
              -2.8122, -6.2512]).reshape(-1, 1)

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o', markersize=4, label='Data')
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.show()
```



Start with Linear Regression

```
A = np.hstack([x**0, x])
A = np.asmatrix(A)

theta = (A.T*A).I*A.T*y
print(theta)
```

```
[[-0.7774     ]
 [-0.71070424]]
```



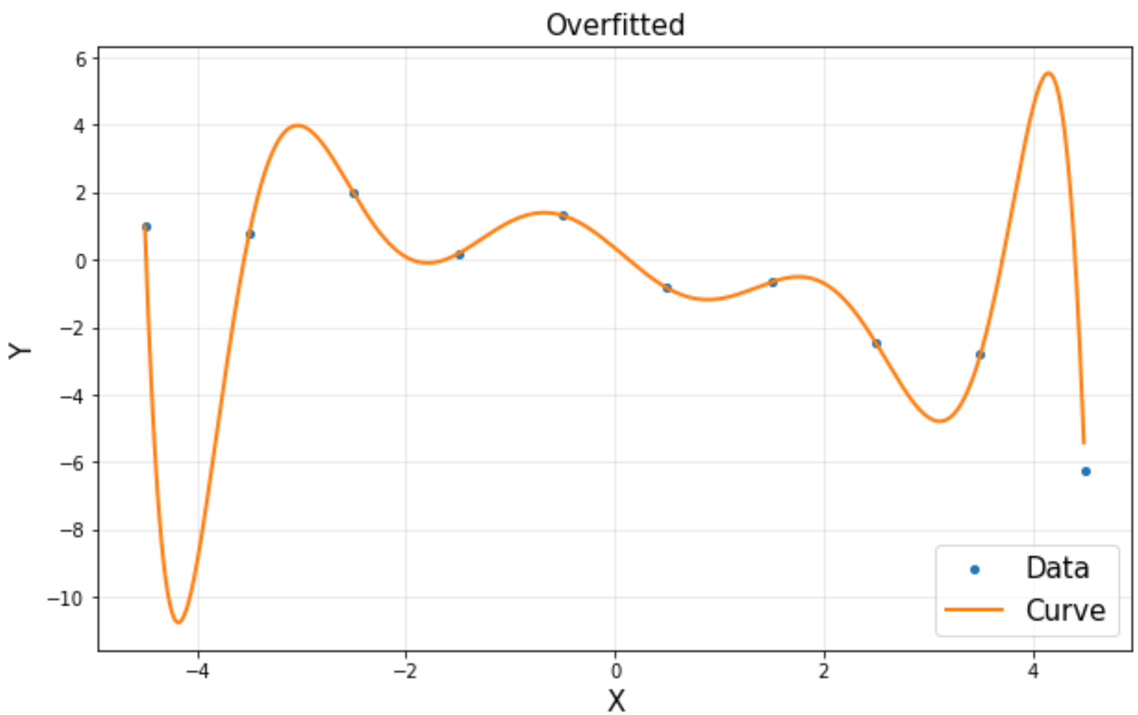
Overfitting

- 10 input points with degree 9 (or 10)

```
A = np.hstack([x**0, x, x**2, x**3, x**4, x**5, x**6, x**7, x**8, x**9])
#A = np.hstack([x**i for i in range(10)])
A = np.asmatrix(A)

theta = (A.T*A).I*A.T*y
print(theta)
```

```
[[  3.48274701e-01]
 [ -2.58951123e+00]
 [ -4.55286474e-01]
 [  1.85022226e+00]
 [  1.06250369e-01]
 [ -4.43328786e-01]
 [ -9.25753472e-03]
 [  3.63088178e-02]
 [  2.35143849e-04]
 [ -9.24099978e-04]]
```

Overfitting



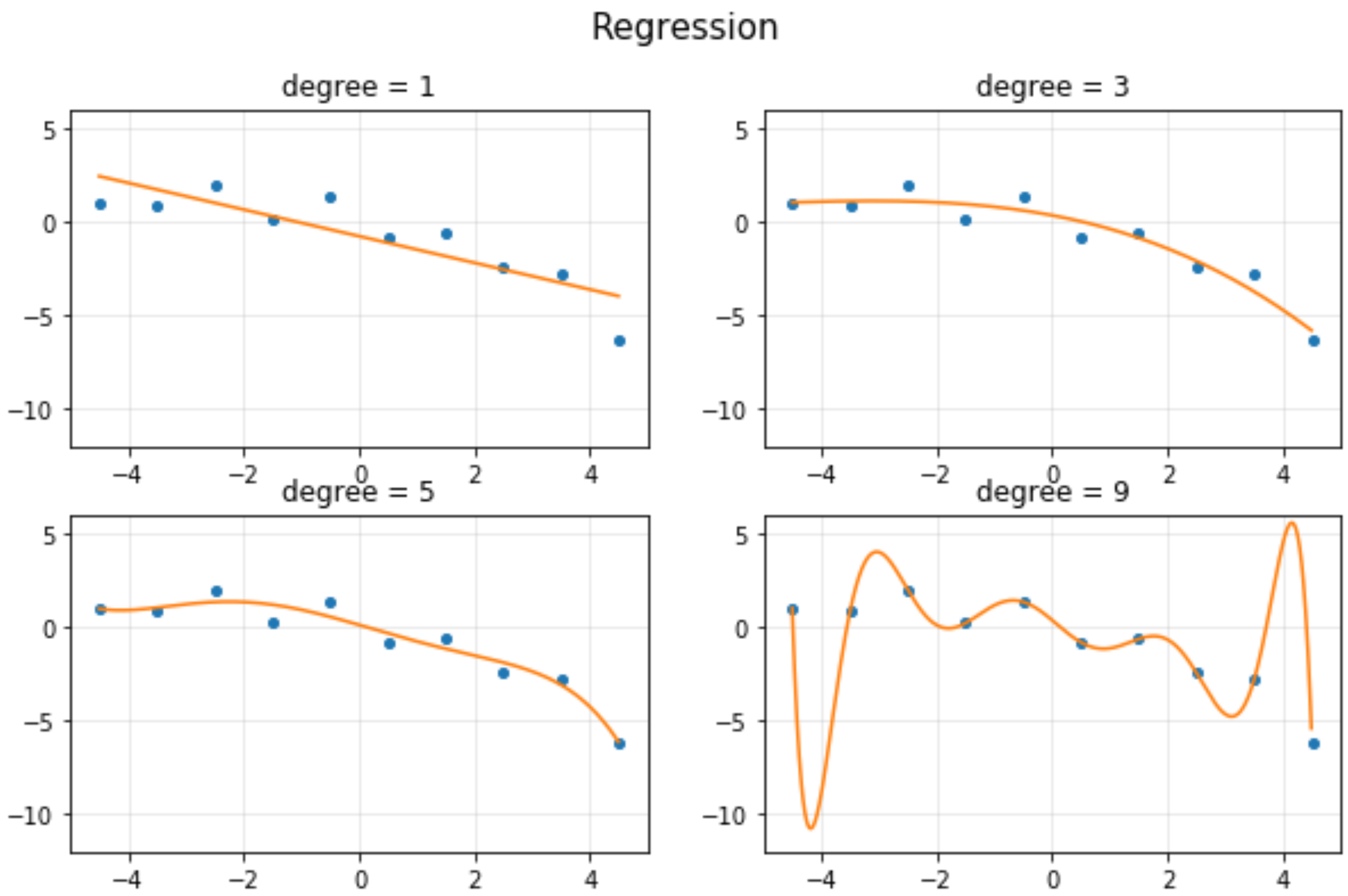
- Low error on input data points, but high error nearby

Polynomial Fitting with Different Degrees

```
d = [1, 3, 5, 9]
RSS = []

plt.figure(figsize=(10, 6))
plt.suptitle('Regression', fontsize=15)

for k in range(4):
    A = np.hstack([x**i for i in range(d[k]+1)])
    polybasis = np.hstack([xp**i for i in range(d[k]+1)])

    A = np.asmatrix(A)
    polybasis = np.asmatrix(polybasis)

    theta = (A.T*A).I*A.T*y
    yp = polybasis*theta

    RSS.append(np.linalg.norm(y - A*theta, 2))

    plt.subplot(2, 2, k+1)
    plt.plot(x, y, 'o', markersize=4)
    plt.plot(xp, yp)
    plt.axis([-5, 5, -12, 6])
    plt.title('degree = {}'.format(d[k]))
    plt.grid(alpha=0.3)

plt.show()
```

Polynomial Fitting with Different Degrees

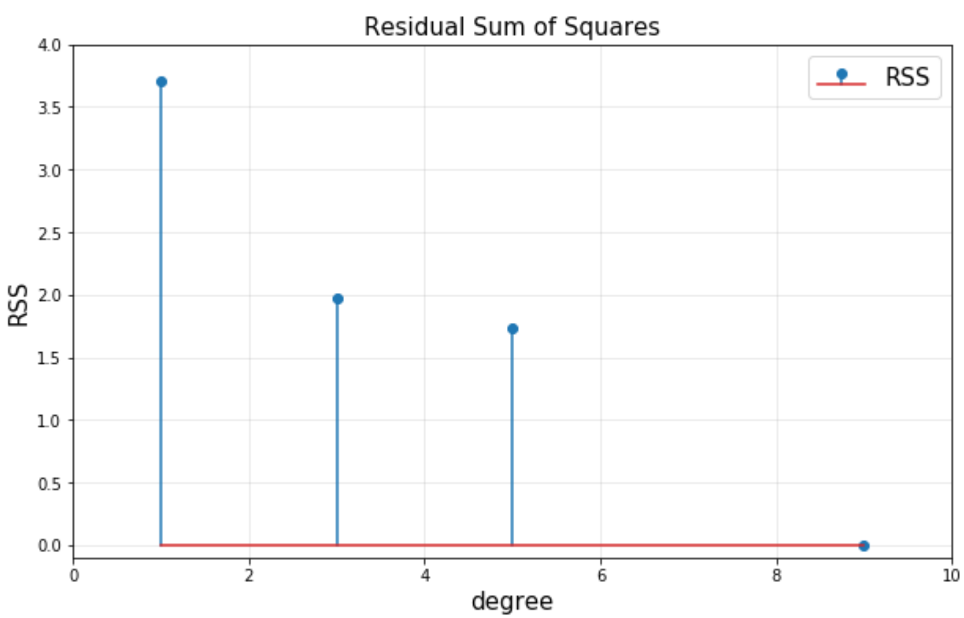
- Least-squares fits for polynomial features of different degrees



Loss

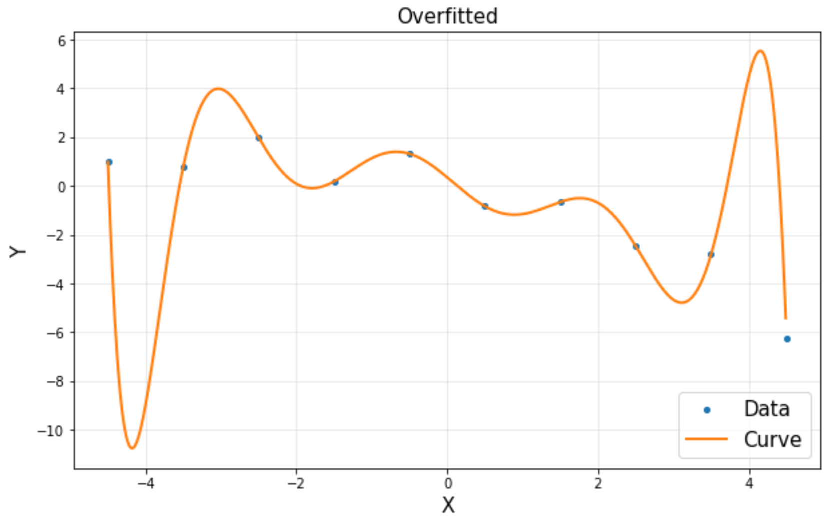
- Loss: Residual Sum of Squares (RSS)

```
plt.figure(figsize=(10, 6))
plt.stem(d, RSS, label='RSS')
plt.title('Residual Sum of Squares', fontsize=15)
plt.xlabel('degree', fontsize=15)
plt.ylabel('RSS', fontsize=15)
plt.axis([0, 10, -0.1, 4.0])
plt.legend(fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```



Issue with Rich Representation

- Low error on input data points, but high error nearby
- Low error on training data, but high error on testing data

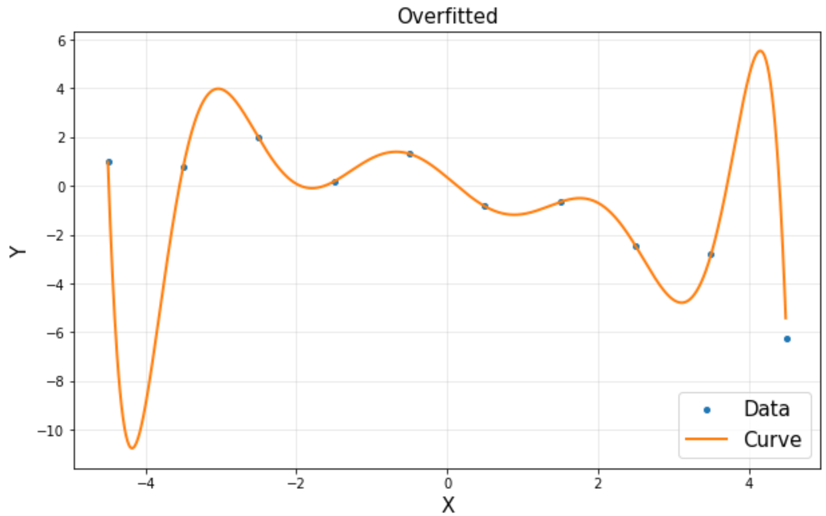


Generalization Error

- Fundamental problem: we are optimizing parameters to solve

$$\min_{\theta} \sum_{i=1}^{m} \ell(y_i, \hat{y}_i) = \min_{\theta} \sum_{i=1}^{m} \ell(y_i, \Phi\theta)$$

- But what we really care about is loss of prediction on new data (x, y)
  - also called generalization error



- Divide data into training set, and validation (testing) set

Representational Difficulties

- With many features, prediction function becomes very expressive (model complexity)
  - Choose less expensive function (e.g., lower degree polynomial, fewer RBF centers, larger RBF bandwidth)
  - Keep the magnitude of the parameter small
  - Regularization: penalize large parameters θ

$$\min \ ||\Phi\theta - y||_2^2 + \lambda||\theta||_2^2$$

  - λ: regularization parameter, trades off between low loss and small values of θ

- We will come back to this issue when talking about regularization

Construct Explicit Feature Vectors

- Consider linear combinations of fixed nonlinear functions of the input variables
  - Polynomial
  - Radial Basis Function (RBF)

$$\hat{y} = \sum_{i=0}^{d} \theta_i \phi_i(x) = \Phi\theta$$

Recap: Nonlinear Regression

- Polynomial (here, quad is used as an example)

$$y = \theta_1 + \theta_2 x + \theta_3 x^2 + \text{noise}$$

$$\phi(x_i) = A = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix}$$

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_m & x_m^2 \end{bmatrix} \implies \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta$$

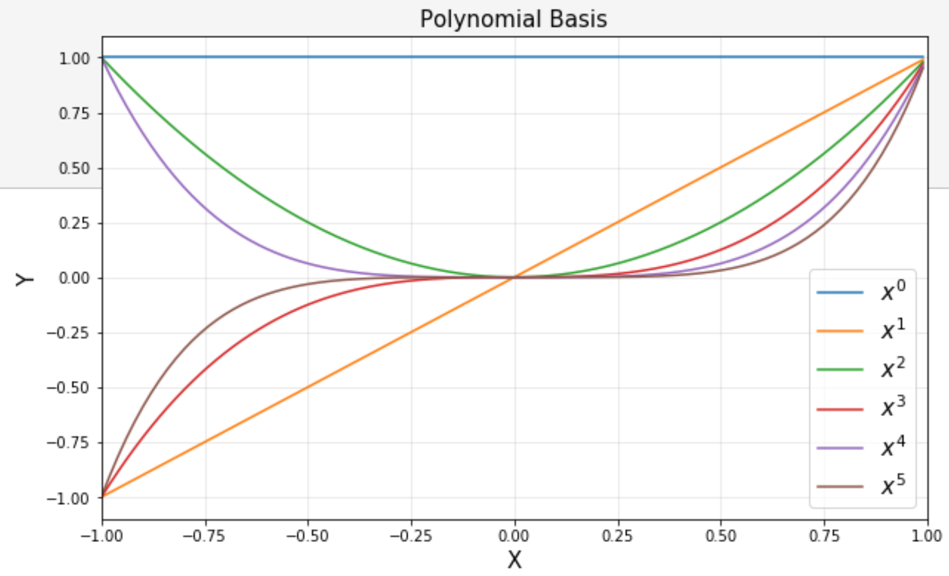
$$\implies \theta^* = (\Phi^T \Phi)^{-1} \Phi^T y$$

Polynomial Basis

1) Polynomial functions

$$\phi_i(x) = x^i, \quad i = 0, \cdots, d$$

```
xp = np.arange(-1, 1, 0.01).reshape(-1, 1)
polybasis = np.hstack([xp**i for i in range(6)])

plt.figure(figsize=(10, 6))

for i in range(6):
    plt.plot(xp, polybasis[:,i], label='$x^{}$'.format(i))

plt.title('Polynomial Basis', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis([-1, 1, -1.1, 1.1])
plt.grid(alpha=0.3)
plt.legend(fontsize=15)
plt.show()
```

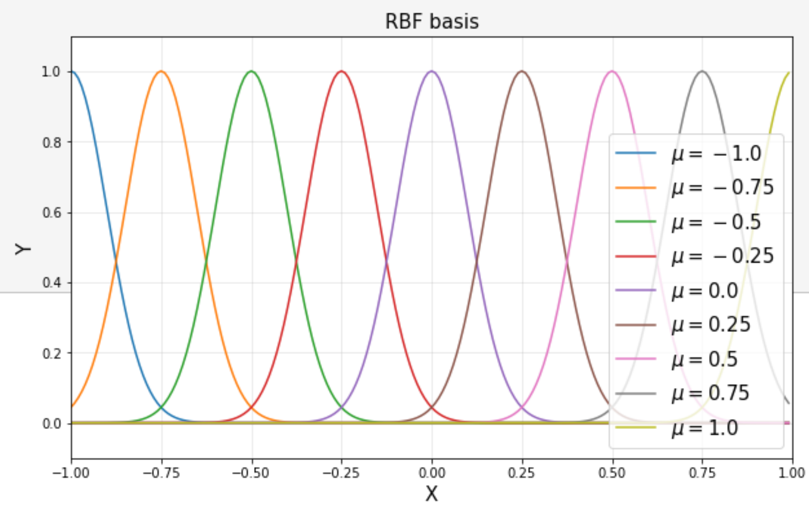


RBF Basis

2) Radial Basis Functions (RBF) with bandwidth σ and k RBF centers $\mu_i \in \mathbb{R}^n$, $i = 1, 2, \cdots, k$

$$\phi_i(x) = \exp\left(-\frac{\|x-\mu_i\|^2}{2\sigma^2}\right)$$

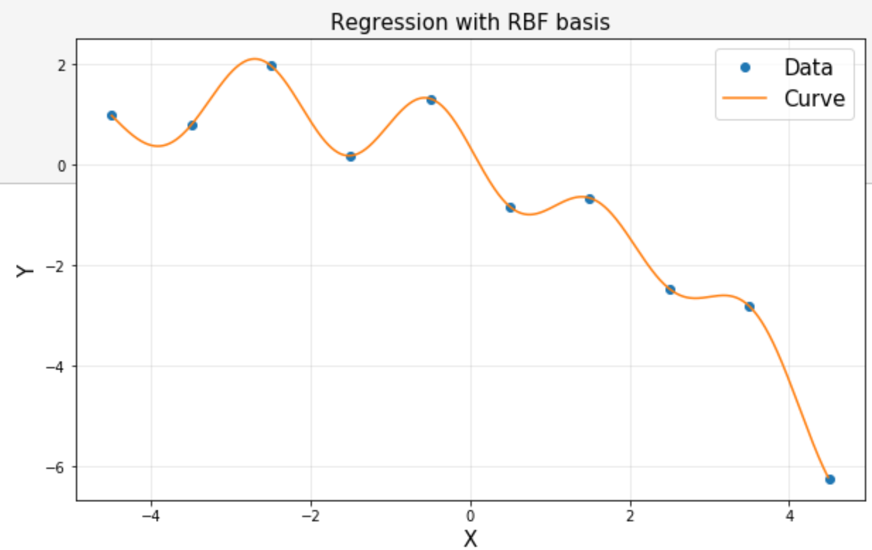
```
d = 9

u = np.linspace(-1, 1, d)
sigma = 0.1

rbfbasis = np.hstack([np.exp(-(xp-u[i])**2/(2*sigma**2)) for i in range(d)])

plt.figure(figsize=(10, 6))

for i in range(d):
    plt.plot(xp, rbfbasis[:,i], label='$\mu = {}$'.format(u[i]))

plt.title('RBF basis', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis([-1, 1, -0.1, 1.1])
plt.legend(loc='lower right', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```



Linear Regression with RBF

```
xp = np.arange(-4.5, 4.5, 0.01).reshape(-1, 1)

d = 10
u = np.linspace(-4.5, 4.5, d)
sigma = 1

A = np.hstack([np.exp(-(x-u[i])**2/(2*sigma**2)) for i in range(d)])
rbfbasis = np.hstack([np.exp(-(xp-u[i])**2/(2*sigma**2)) for i in range(d)])

A = np.asmatrix(A)
rbfbasis = np.asmatrix(rbfbasis)

theta = (A.T*A).I*A.T*y
yp = rbfbasis*theta

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o', label='Data')
plt.plot(xp, yp, label='Curve')
plt.title('Regression with RBF basis', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.grid(alpha=0.3)
plt.legend(fontsize=15)
plt.show()
```

$$\boxed{\theta = (A^T A)^{-1} A^T y}$$

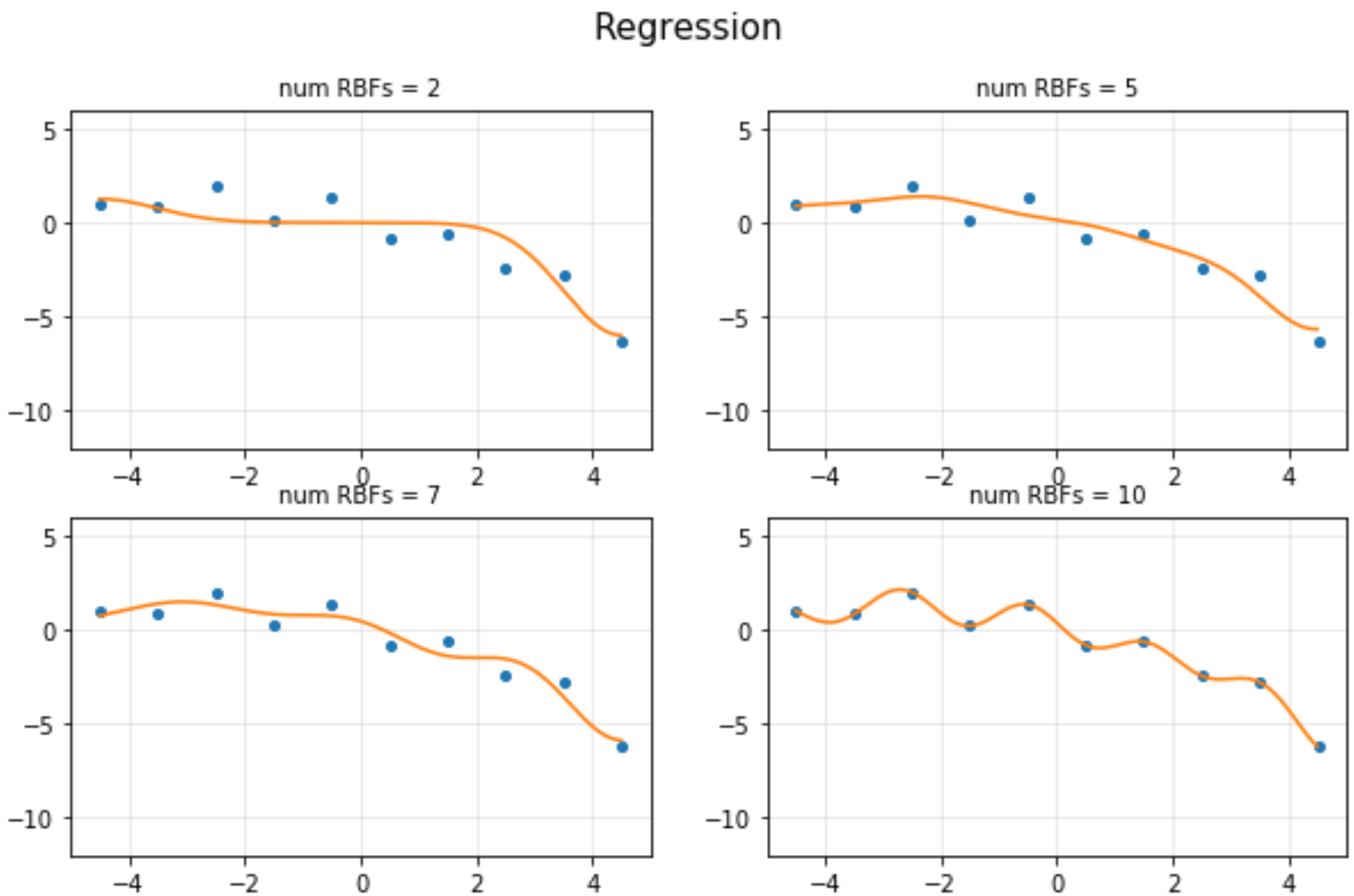


Fewer RBF Centers

```
d = [2, 5, 7, 10]
sigma = 1

plt.figure(figsize=(10, 6))

for k in range(4):
    u = np.linspace(-4.5, 4.5, d[k])

    A = np.hstack([np.exp(-(x-u[i])**2/(2*sigma**2)) for i in range(d[k])])
    rbfbasis = np.hstack([np.exp(-(xp-u[i])**2/(2*sigma**2)) for i in range(d[k])])

    A = np.asmatrix(A)
    rbfbasis = np.asmatrix(rbfbasis)

    theta = (A.T*A).I*A.T*y
    yp = rbfbasis*theta

    plt.subplot(2, 2, k+1)
    plt.plot(x, y, 'o', markersize=4)
    plt.plot(xp, yp)
    plt.axis([-5, 5, -12, 6])
    plt.title('num RBFs = {}'.format(d[k]), fontsize=10)
    plt.grid(alpha=0.3)

plt.suptitle('Regression', fontsize=15)
plt.show()
```

Fewer RBF Centers

- Least-squares fits for different numbers of RBFs



Regularization (Shrinkage Methods)

- Often, overfitting associated with very large estimated parameters
- We want to balance
  - how well function fits data
  - magnitude of coefficients

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot \|\theta\|_2^2}$$

$$\implies \min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

  - multi-objective optimization
  - λ is a tuning parameter

– the second term, $\lambda \cdot \|\theta\|_2^2$, called a shrinkage penalty, is small when $\theta_1, \cdots, \theta_d$ are close to zeros, and so it has the effect of shrinking the estimates of θ_j towards zero

– the tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates

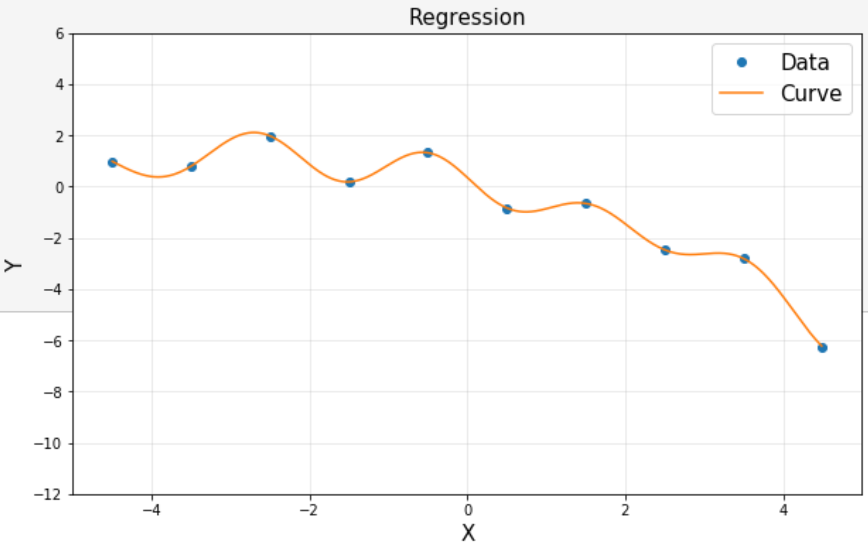
– known as a *ridge regression*

$$\min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

RBF

```
# CVXPY code

d = 10
u = np.linspace(-4.5, 4.5, d)

sigma = 1

A = np.hstack([np.exp(-(x-u[i])**2/(2*sigma**2)) for i in range(d)])
rbfbasis = np.hstack([np.exp(-(xp-u[i])**2/(2*sigma**2)) for i in range(d)])

theta = cvx.Variable(d, 1)
obj = cvx.Minimize(cvx.norm(A*theta-y, 2))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o', label='Data')
plt.plot(xp, yp, label='Curve')
plt.title('Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis([-5, 5, -12, 6])
plt.legend(fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```

$$\boxed{\min \|\Phi\theta - y\|_2^2}$$



RBF with Regularization

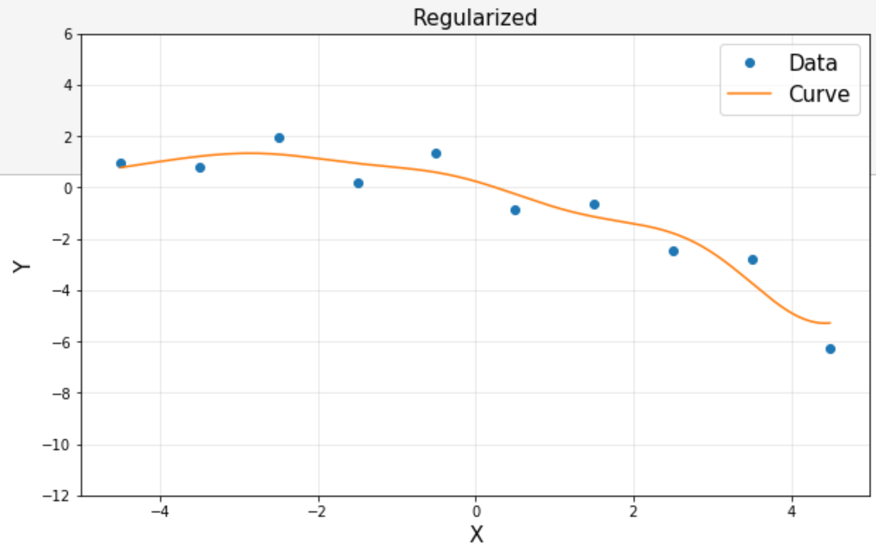
```
# ridge regression

A = np.hstack([np.exp(-(x-u[i])**2/(2*sigma**2)) for i in range(d)])
rbfbasis = np.hstack([np.exp(-(xp-u[i])**2/(2*sigma**2)) for i in range(d)])

lamb = 0.1
theta = cvx.Variable(d, 1)
obj = cvx.Minimize(cvx.sum_squares(A*theta - y) + lamb*cvx.sum_squares(theta))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o', label='Data')
plt.plot(xp, yp, label='Curve')
plt.title('Regularized', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis([-5, 5, -12, 6])
plt.legend(fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```

$$\boxed{\min \; \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2}$$

RBF with Regularization

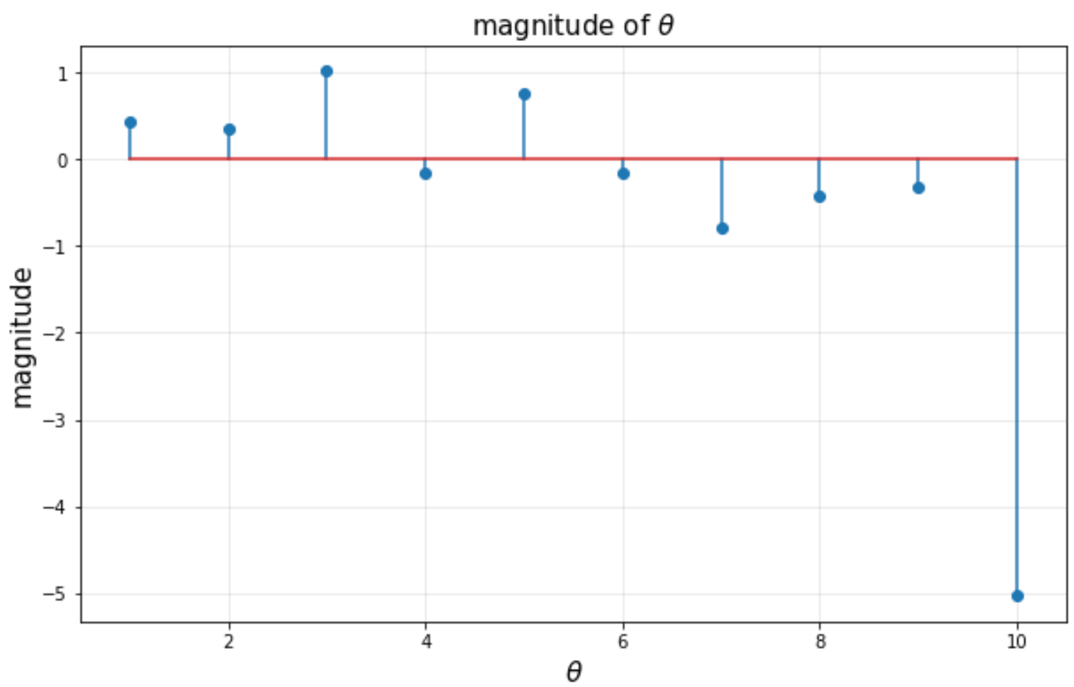
```
# ridge regression

A = np.hstack([np.exp(-(x-u[i])**2/(2*sigma**2)) for i in range(d)])
rbfbasis = np.hstack([np.exp(-(xp-u[i])**2/(2*sigma**2)) for i in range(d)])

lamb = 0.1
theta = cvx.Variable(d, 1)
obj = cvx.Minimize(cvx.sum_squares(A*theta - y) + lamb*cvx.sum_squares(theta))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'o', label='Data')
plt.plot(xp, yp, label='Curve')
plt.title('Regularized', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis([-5, 5, -12, 6])
plt.legend(fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```

$$\boxed{\min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2}$$



Weights

```
# Regulization ( = ridge nonlinear regression) encourages small weights, but not exactly 0
plt.figure(figsize=(10, 6))
plt.title(r'magnitude of $\theta$', fontsize=15)
plt.xlabel(r'$\theta$', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.stem(np.linspace(1, 10, 10).reshape(-1, 1), theta.value)
plt.xlim([0.5, 10.5])
plt.grid(alpha=0.3)
plt.show()
```



Let's Use L_1 Norm

- Try this cost instead of ridge...

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot \|\theta\|_1}$$

$$\implies \min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1$$

- λ is a tuning parameter = balance of fit and sparsity
- Known as *Lasso*
  – least absolute shrinkage and selection operator

RBF with Lasso

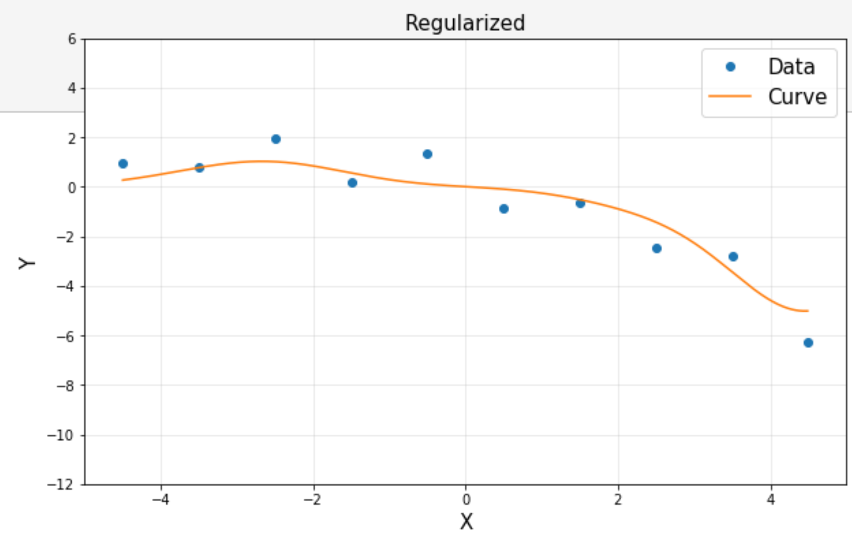
```
# LASSO regression

lamb = 2
theta = cvx.Variable(d, 1)
obj = cvx.Minimize(cvx.sum_squares(A*theta - y) + lamb*cvx.norm(theta, 1))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value

plt.figure(figsize=(10, 6))
plt.title('Regularized', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'o', label='Data')
plt.plot(xp, yp, label='Curve')
plt.axis([-5, 5, -12, 6])
plt.legend(fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```

$$\boxed{\min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1}$$

RBF with Lasso

```
# LASSO regression

lamb = 2
theta = cvx.Variable(d, 1)
obj = cvx.Minimize(cvx.sum_squares(A*theta - y) + lamb*cvx.norm(theta, 1))
prob = cvx.Problem(obj).solve()

yp = rbfbasis*theta.value

plt.figure(figsize=(10, 6))
plt.title('Regularized', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'o', label='Data')
plt.plot(xp, yp, label='Curve')
plt.axis([-5, 5, -12, 6])
plt.legend(fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```

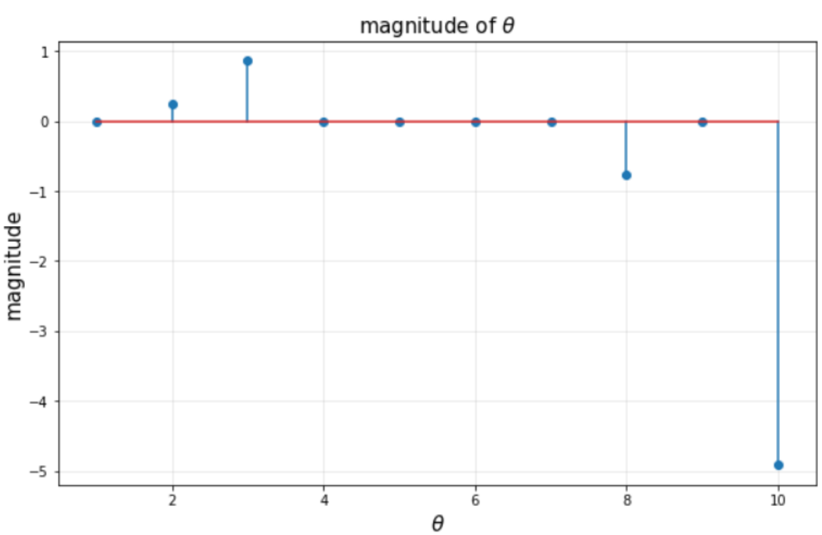
$$\min \; \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1$$



Weights with Lasso

- Non-zero coefficients indicate ‘selected’ features

```
# Regulization ( = ridge nonlinear regression) encourages small weights, but not exactly 0
plt.figure(figsize=(10, 6))
plt.title(r'magnitude of $\theta$', fontsize=15)
plt.xlabel(r'$\theta$', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.stem(np.linspace(1, 10, 10).reshape(-1, 1), theta.value)
plt.xlim([0.5, 10.5])
plt.grid(alpha=0.3)
plt.show()
```

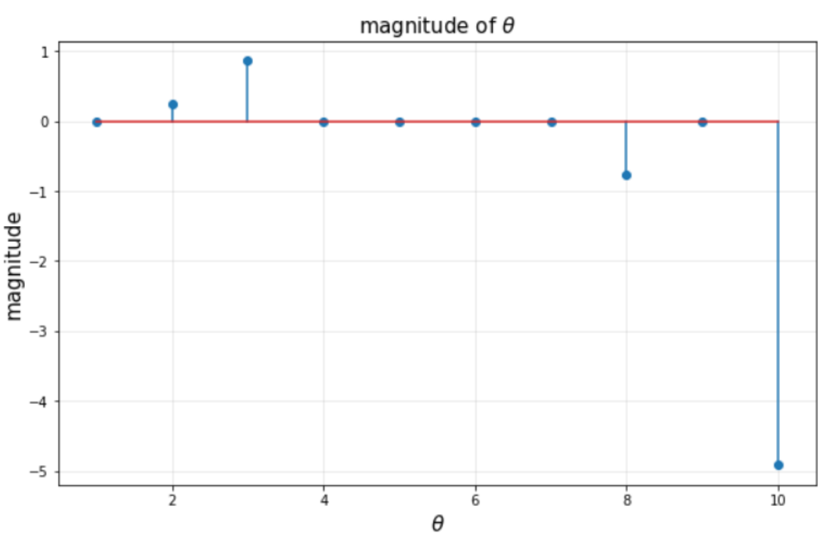


Lasso

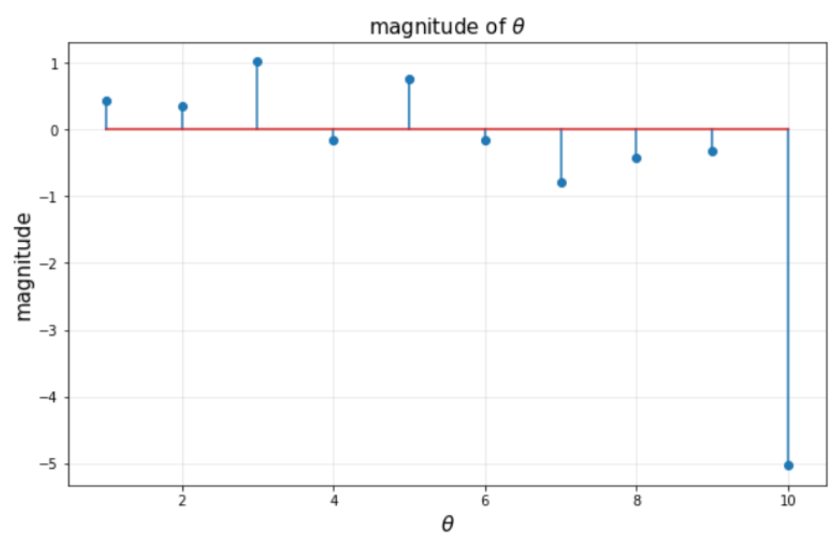
Weights with Lasso

- Non-zero coefficients indicate ‘selected’ features

```
# Regulization ( = ridge nonlinear regression) encourages small weights, but not exactly 0
plt.figure(figsize=(10, 6))
plt.title(r'magnitude of $\theta$', fontsize=15)
plt.xlabel(r'$\theta$', fontsize=15)
plt.ylabel('magnitude', fontsize=15)
plt.stem(np.linspace(1, 10, 10).reshape(-1, 1), theta.value)
plt.xlim([0.5, 10.5])
plt.grid(alpha=0.3)
plt.show()
```



Lasso



Ridge

Sparsity for Feature Selection using Lasso

- Least squares with a penalty on the L_1 norm of the parameters

- Start with full model (all possible features)

- 'Shrink' some coefficients exactly to 0
  - *i.e.*, knock out certain features
  - The L_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero

- Non-zero coefficients indicate 'selected' features

Lasso vs. Ridge

- Another equivalent forms of optimizations

$$\min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1$$

$$\min \ \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

$$\begin{array}{ll} \displaystyle\min_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_1 \leq s \end{array}$$

$$\begin{array}{ll} \min_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_2^2 \leq s \end{array}$$

Lasso vs. Ridge

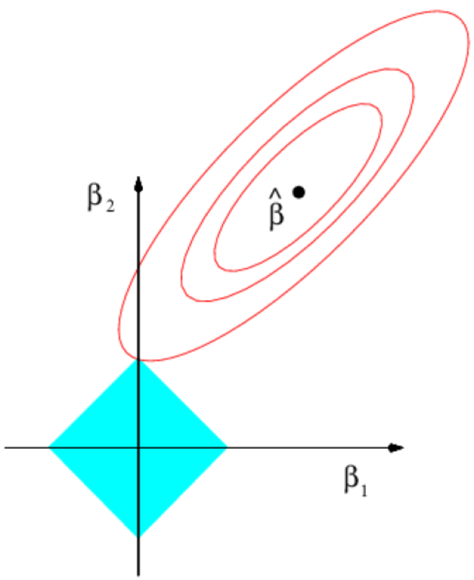
- Another equivalent forms of optimizations

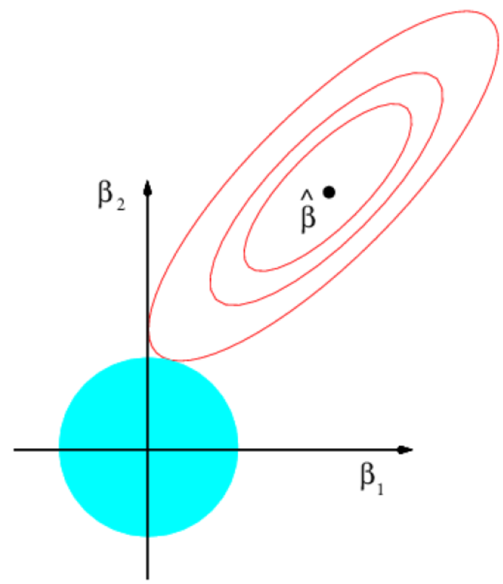
$$\min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_1$$

$$\min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

$$\begin{array}{ll} \min\limits_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_1 \leq s \end{array}$$

$$\begin{array}{ll} \min_{\theta} & \|\Phi\theta - y\|_2^2 \\ \text{subject to} & \|\theta\|_2^2 \leq s \end{array}$$

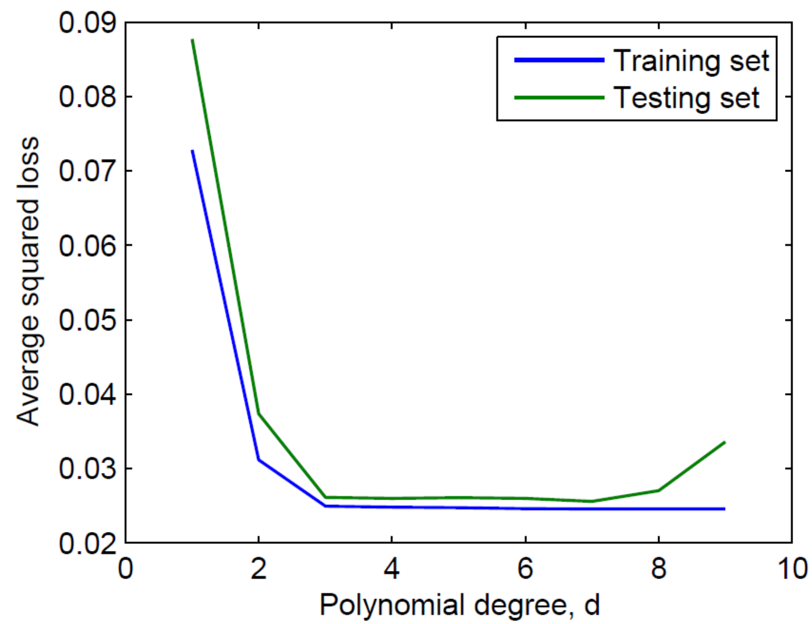




Evaluation

- Adding more features will always decrease the loss
- How do we determine when an algorithm achieves "good" performance?

- A better criterion:
  - Training set (e.g., 70 %)
  - Testing set (e.g., 30 %)



Testing loss versus degree of polynomial

- Performance on testing set called *generalization* performance