Linear Regression

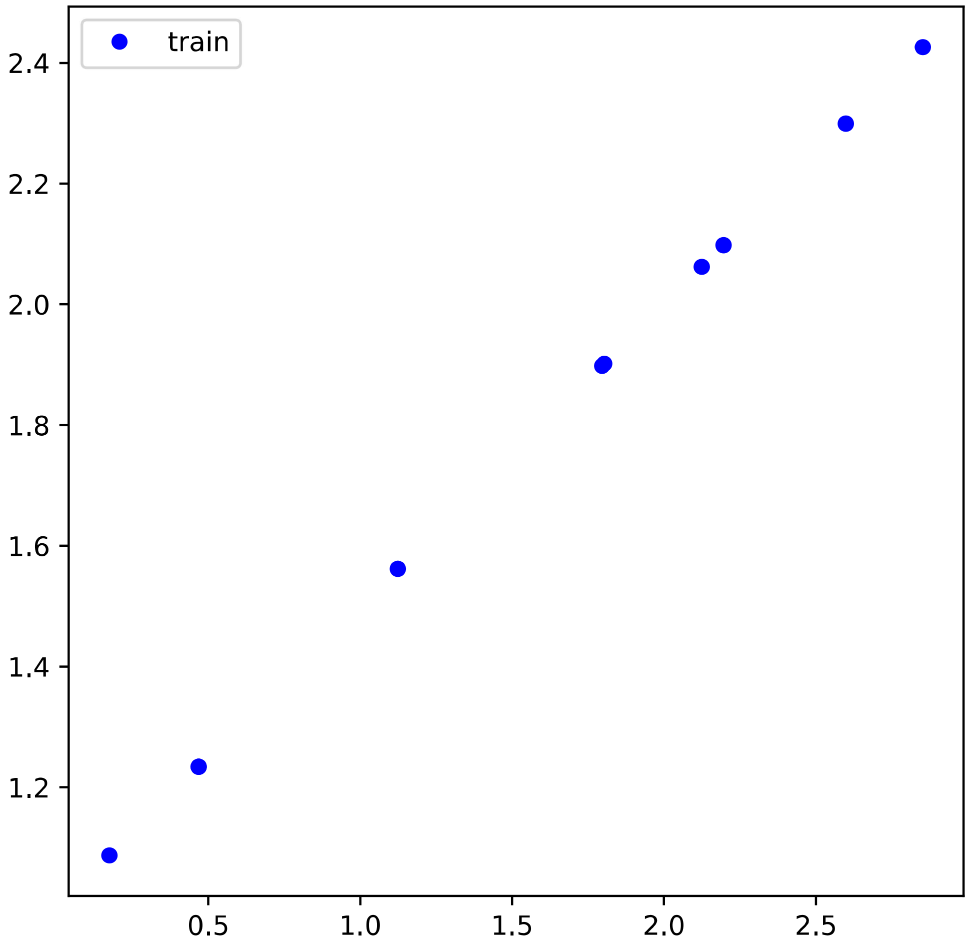
In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures # "Feature Engineering" for
from sklearn.metrics import mean_squared_error
```

In [2]:
```
%matplotlib inline
%config InlineBackend.figure_formats = ['svg']
```

Assume that the (unknown) target function is

$$y = f(x) = 1.0 + 0.5 \times x$$

In [3]:
```
# Create some training data item
np.random.seed(42)
N = 10
X_train = 3.0*np.random.rand(N, 1) # rand(N,1): Create N random numbers in the i
y_train = 1.0 + 0.5*X_train  # randn(N,1): Create N random numbers ~ standard no
plt.figure(figsize=(6,6))
plt.plot(X_train, y_train, 'b.', markersize=10, label='train')
plt.legend()
plt.show()
```



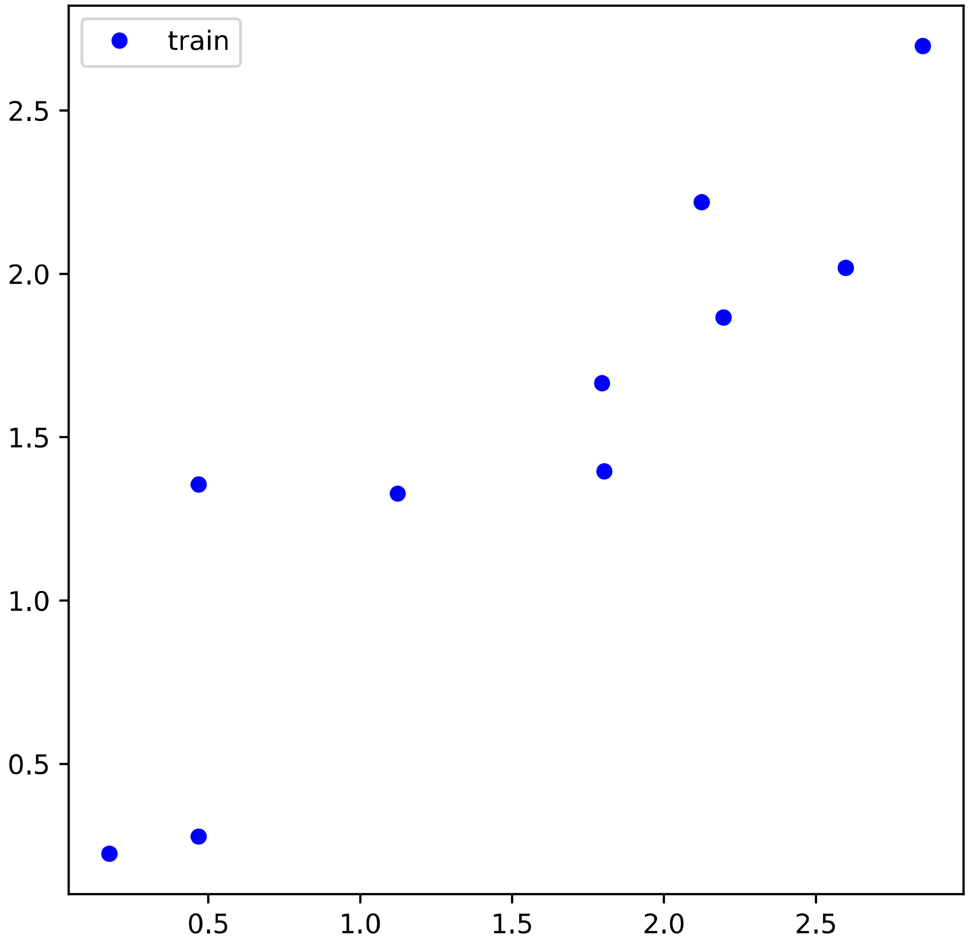
The data items generated above don't have any noise, so they all lie on a straight line.

To simulate the fact that data collected in practice usually contain certain noises, we add some noises to the generated target values.

$y = f(x) = 1.0 + 0.5 \times x + \varepsilon$

In [4]:
```
# Create some training data item
np.random.seed(42)
N = 10
X_train = 3.0*np.random.rand(N, 1) # rand(N,1): Create N random numbers in the i
y_train = 1.0 + 0.5*X_train + np.random.randn(N,1)/2.0  # randn(N,1): Create N r
```

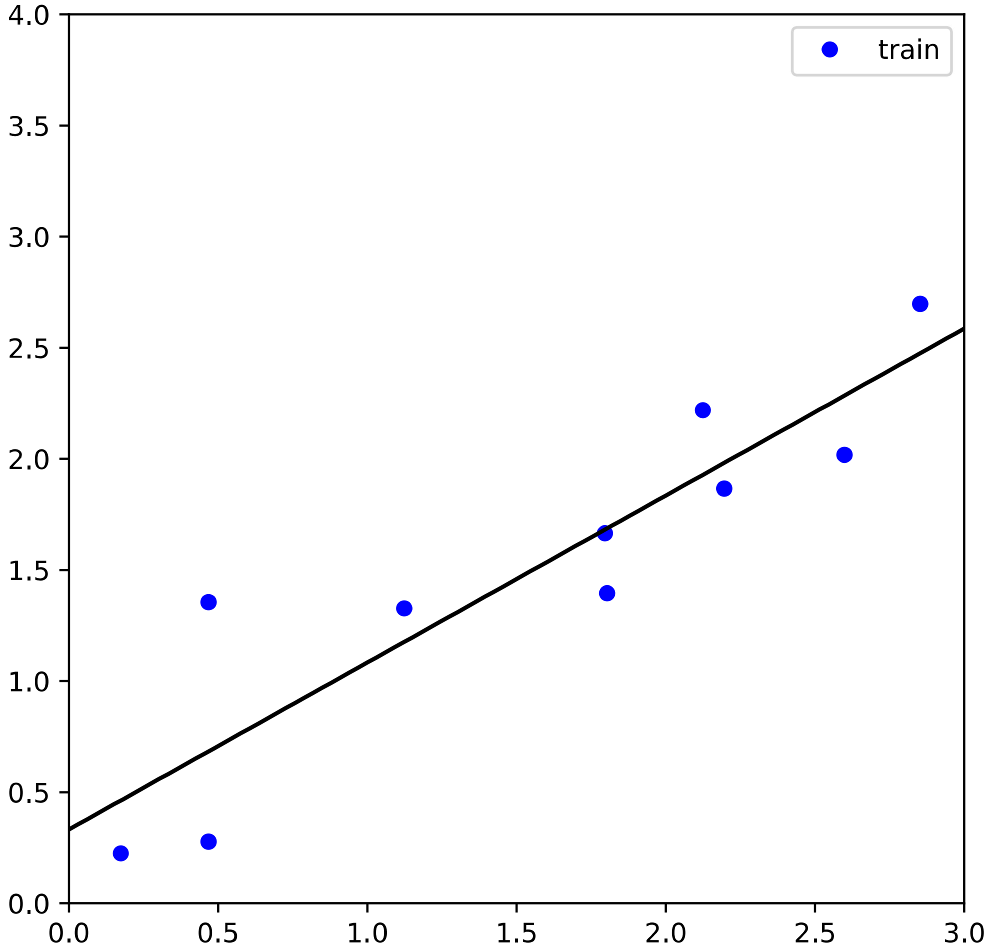
In [5]:
```
plt.figure(figsize=(6,6))
plt.plot(X_train, y_train, 'b.', markersize=10, label='train')
#plt.plot(X_test, y_test, 'r*', markersize=10, label='test')
plt.legend()
plt.show()
```



In [6]:
```
# Create some data to use for plotting
X_new = np.linspace(0,3,100).reshape(100,1) # Create an array of 100 equal-space
```

In [7]:
```
lin_reg = LinearRegression() # Normal Equation
lin_reg.fit(X_train,y_train)

plt.figure(figsize=(6,6))
plt.plot(X_train,y_train,'b.', ms=10, label='train')
plt.plot(X_new, lin_reg.predict(X_new), 'k-')
```

```
plt.legend()
plt.axis([0, 3, 0, 4])
plt.show()
print(lin_reg.intercept_, lin_reg.coef_)  # intercept_: w0 (sometimes we call it
# Here we only have a simple linear regression model: predict = w0 + w1*x

y_pred_train = lin_reg.predict(X_train) # Use the trained model to make predicti
lr_loss_train = mean_squared_error(y_train, y_pred_train) # Mean Squared Error L
print('Training Loss: ', lr_loss_train)
```




```
[0.33181066] [[0.75155622]]
Training Loss:  0.09991531905633781
```

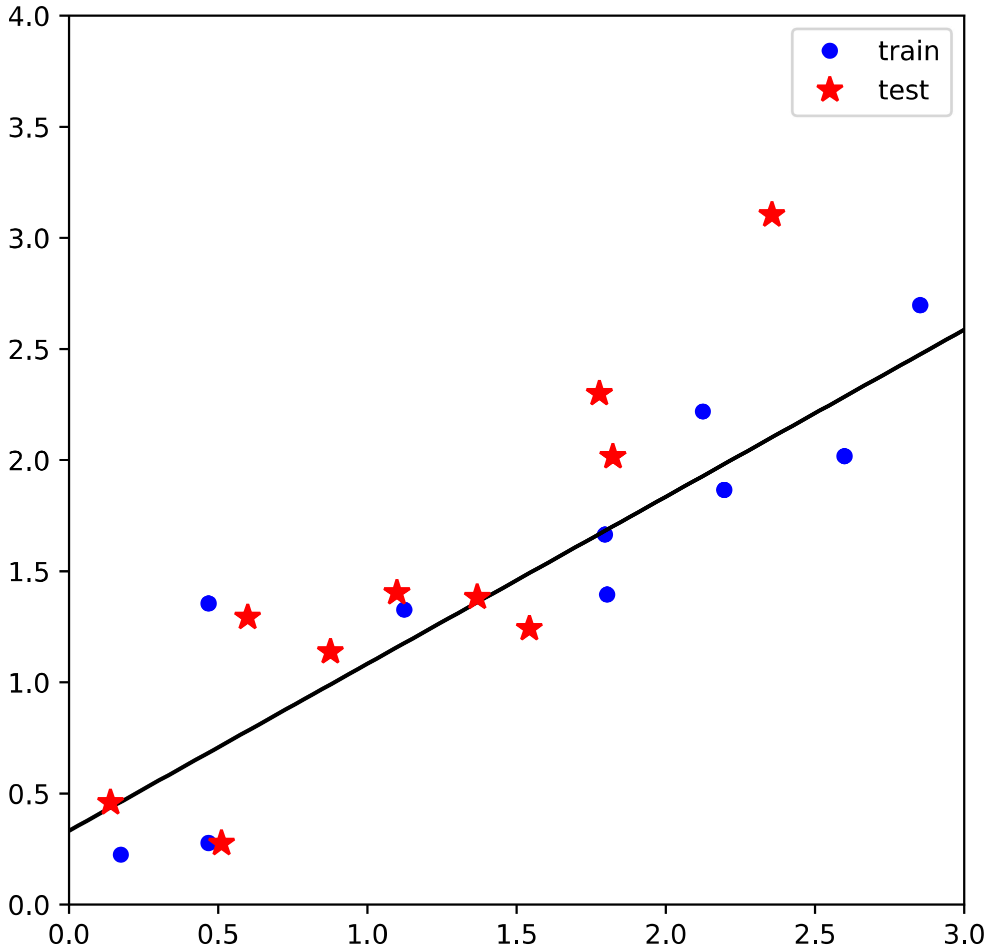
In [8]:
```
print(X_train)
print(y_train)
```

```
[[1.12362036]
 [2.85214292]
 [2.19598183]
 [1.79597545]
 [0.46805592]
 [0.46798356]
 [0.17425084]
 [2.59852844]
 [1.80334504]
 [2.12421773]]
[[1.32707299]
 [2.69735148]
 [1.86628207]
 [1.66512285]
 [1.3550091 ]
 [0.27735166]
 [0.2246665 ]
 [2.01812045]
 [1.39525696]
 [2.21923253]]
```

After training, we would like to evaluate the trained model on "test" data.

In [9]:
```
# Create some test data
N_test = 10
X_test = 3.0*np.random.rand(N_test, 1)
y_test = 1.0 + 0.5*X_test + np.random.randn(N_test,1)/2.0 # Note: Testing datase
```

In [10]:
```
plt.figure(figsize=(6,6))
plt.plot(X_train, y_train,'b.', ms=10, label='train')
plt.plot(X_new, lin_reg.predict(X_new), 'k-')
plt.plot(X_test, y_test, 'r*', ms=10, label='test')
plt.legend()
plt.axis([0, 3, 0, 4])
plt.show()

y_pred_train = lin_reg.predict(X_train) # Use the trained model to make predicti
lr_loss_train = mean_squared_error(y_train, y_pred_train) # Mean Squared Error L
print('Training Loss: ', lr_loss_train)

y_pred_test = lin_reg.predict(X_test) # Use the trained model to make prediction
lr_loss_test = mean_squared_error(y_test, y_pred_test) # Mean Squared Error Loss
print('Testing Loss: ', lr_loss_test)
```



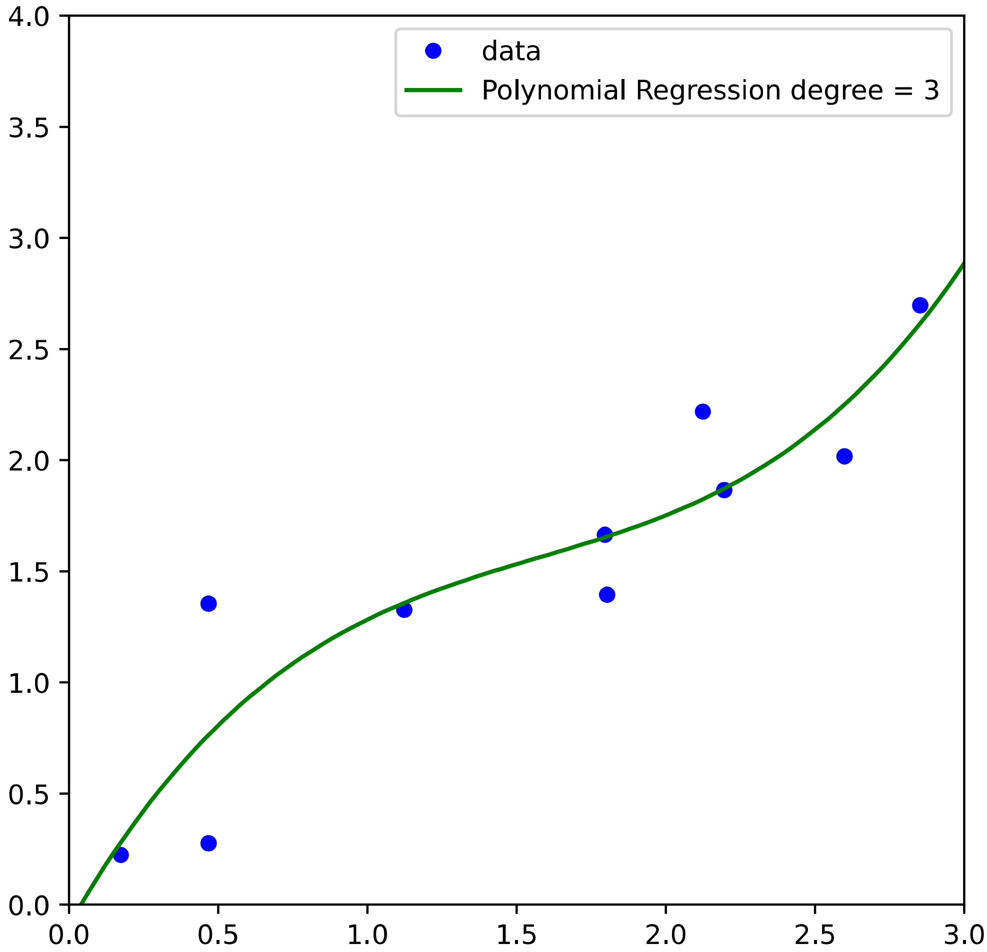

```
Training Loss:  0.09991531905633781
Testing Loss:  0.21020128681013045
```

Polynomial Regression

$$\texttt{prediction} = w_0 + w_1x + w_2x^2 + w_3x^3 + \ldots + w_Dx^D$$

In [11]:
```
deg = 3
poly = PolynomialFeatures(degree=deg, include_bias=False)
X_poly = poly.fit_transform(X_train) # x --> x, x^2, x^3
X_new_poly = poly.fit_transform(X_new) # x --> x, x^2, x^3

poly_reg = LinearRegression()  # In essence, Polynomial Regression is still Line
poly_reg.fit(X_poly,y_train)

plt.figure(figsize=(6,6))
plt.plot(X_train, y_train, 'b.', ms=10, label='data')
# plt.plot(X_new, lin_reg.predict(X_new), 'k-', label='Linear Regression')
# print(lin_reg.intercept_, lin_reg.coef_)

plt.plot(X_new, poly_reg.predict(X_new_poly), 'g', label='Polynomial Regression
print(poly_reg.intercept_, poly_reg.coef_)
plt.legend()
plt.axis([0, 3, 0, 4])
plt.show()
poly_loss = mean_squared_error(y_train, poly_reg.predict(X_poly))
print('Training Loss: ', poly_loss)
```

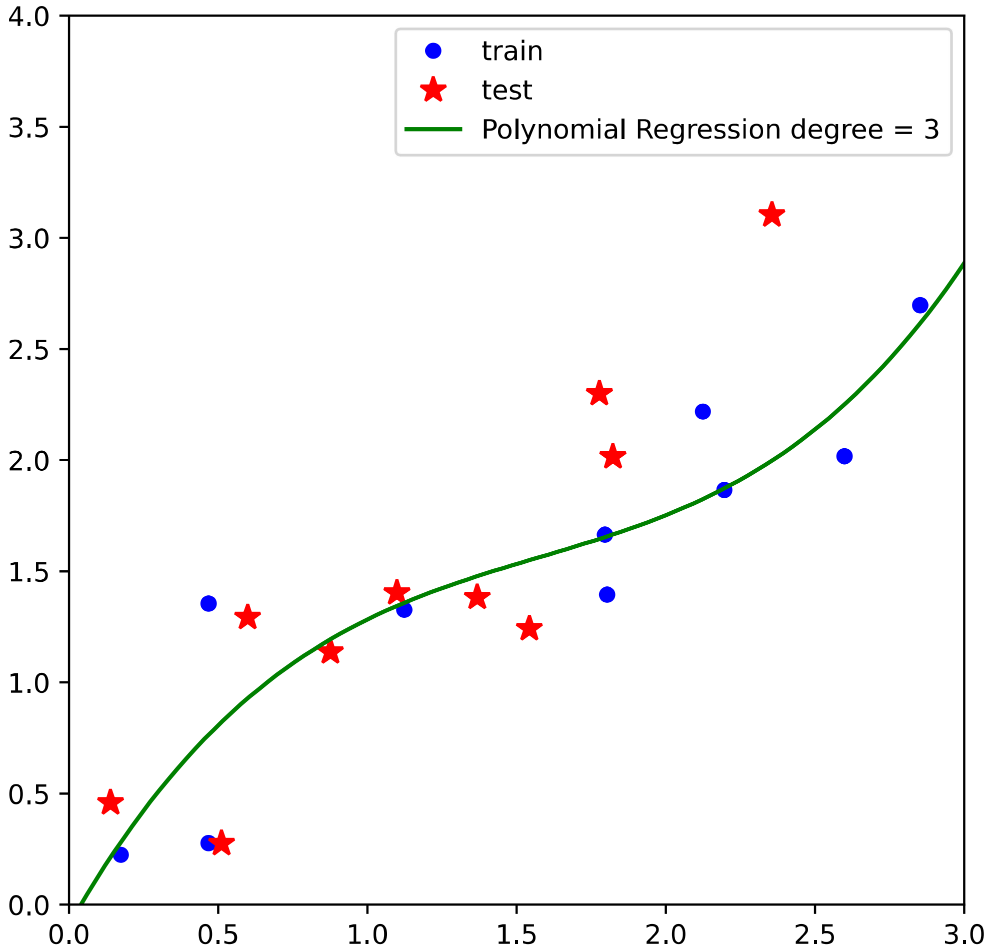
```
[-0.0916451] [[ 2.35120734 -1.23811156  0.26177074]]
```




```
Training Loss:  0.0875019006667126
```

In [12]:
```
plt.figure(figsize=(6,6))
plt.plot(X_train, y_train,'b.', ms=10, label='train')
plt.plot(X_test,y_test,'r*', ms=10, label='test')
# plt.plot(X_new, lin_reg.predict(X_new), 'k-', label='Linear Regression')
# print(lin_reg.intercept_, lin_reg.coef_)

plt.plot(X_new, poly_reg.predict(X_new_poly), 'g', label='Polynomial Regression
print(poly_reg.intercept_, poly_reg.coef_)
plt.legend()
plt.axis([0, 3, 0, 4])
plt.show()

poly_loss_train = mean_squared_error(y_train, poly_reg.predict(X_poly))
print('Training Loss: ', poly_loss_train)
X_test_poly = poly.fit_transform(X_test)
poly_loss_test = mean_squared_error(y_test, poly_reg.predict(X_test_poly))
print('Testing Loss: ', poly_loss_test)
```

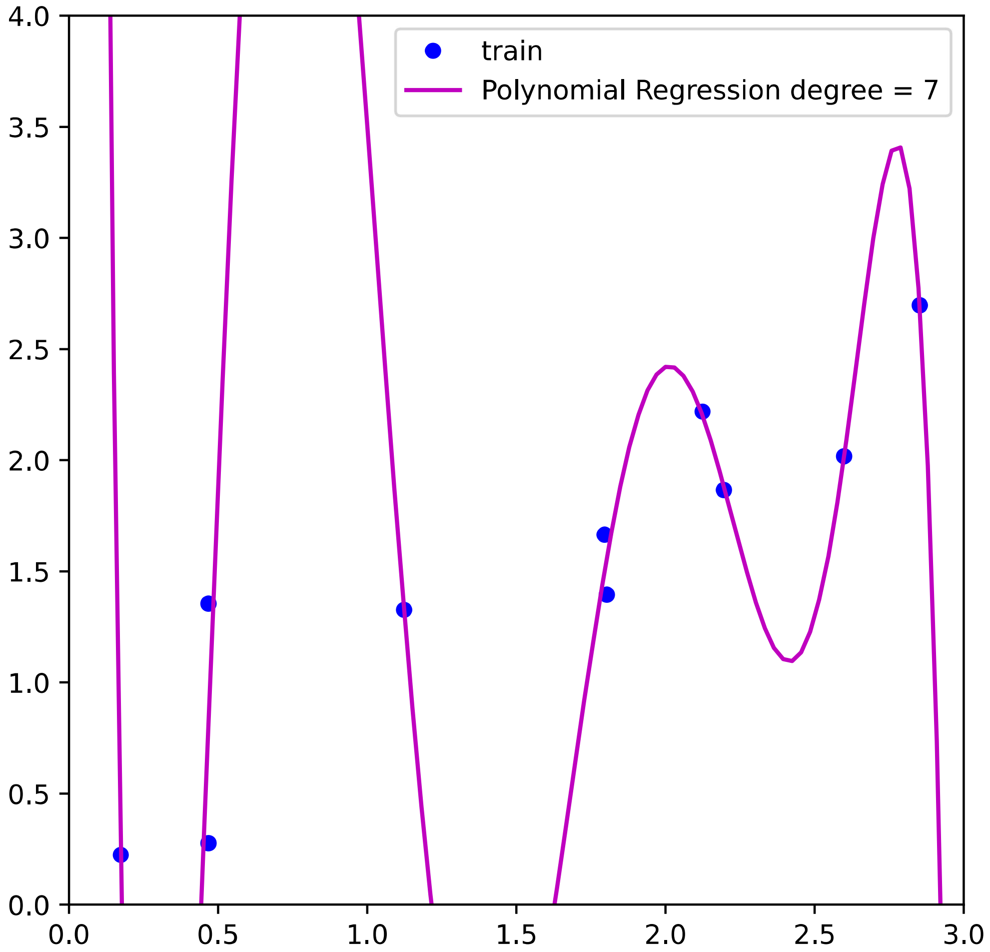
```
[-0.0916451] [[ 2.35120734 -1.23811156  0.26177074]]
```




```
Training Loss:  0.0875019006667126
Testing Loss:  0.23747048298743464
```

In [13]:
```
deg = 7
poly = PolynomialFeatures(degree=deg, include_bias=False)
X_poly = poly.fit_transform(X_train) # x --> x, x^2, x^3, ..., x^7
X_new_poly = poly.fit_transform(X_new) # x --> x, x^2, x^3, ..., x^7

poly_reg = LinearRegression()  # In essence, Polynomial Regression is still Line
poly_reg.fit(X_poly,y_train)

plt.figure(figsize=(6,6))
plt.plot(X_train, y_train,'b.', ms=10, label='train')
# plt.plot(X_new, lin_reg.predict(X_new), 'k-', label='Linear Regression')
# print(lin_reg.intercept_, lin_reg.coef_)

plt.plot(X_new, poly_reg.predict(X_new_poly), 'm', label='Polynomial Regression
print(poly_reg.intercept_, poly_reg.coef_)
plt.legend()
plt.axis([0, 3, 0, 4])
plt.show()
poly_loss_train = mean_squared_error(y_train, poly_reg.predict(X_poly))
print('Training Loss: ', poly_loss_train)
```

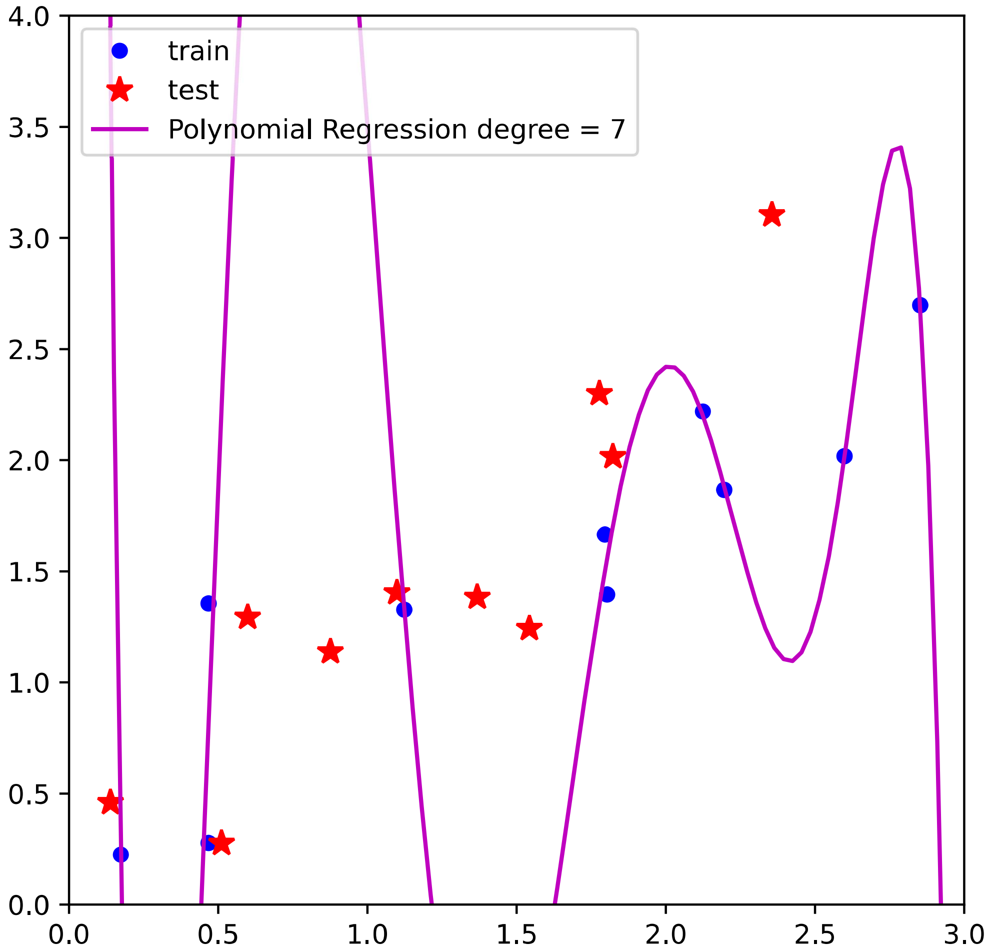
```
[38.069871] [[ -399.60873887  1364.69579468 -2093.21939506  1665.82047777
   -716.24860915   158.03695709   -14.0390417 ]]
```




```
Training Loss:  0.06322418953679351
```

In [14]:
```
plt.figure(figsize=(6,6))
plt.plot(X_train, y_train,'b.', ms=10, label='train')
plt.plot(X_test,y_test,'r*', ms=10, label='test')
# plt.plot(X_new, lin_reg.predict(X_new), 'k-', label='Linear Regression')
# print(lin_reg.intercept_, lin_reg.coef_)

plt.plot(X_new, poly_reg.predict(X_new_poly), 'm', label='Polynomial Regression
print(poly_reg.intercept_, poly_reg.coef_)
plt.legend()
plt.axis([0, 3, 0, 4])
plt.show()

poly_loss_train = mean_squared_error(y_train, poly_reg.predict(X_poly))
print('Training Loss: ', poly_loss_train)
X_test_poly = poly.fit_transform(X_test)
poly_loss_test = mean_squared_error(y_test, poly_reg.predict(X_test_poly))
print('Testing Loss: ', poly_loss_test)
```

```
[38.069871] [[ -399.60873887  1364.69579468 -2093.21939506  1665.82047777
   -716.24860915   158.03695709   -14.0390417 ]]
```



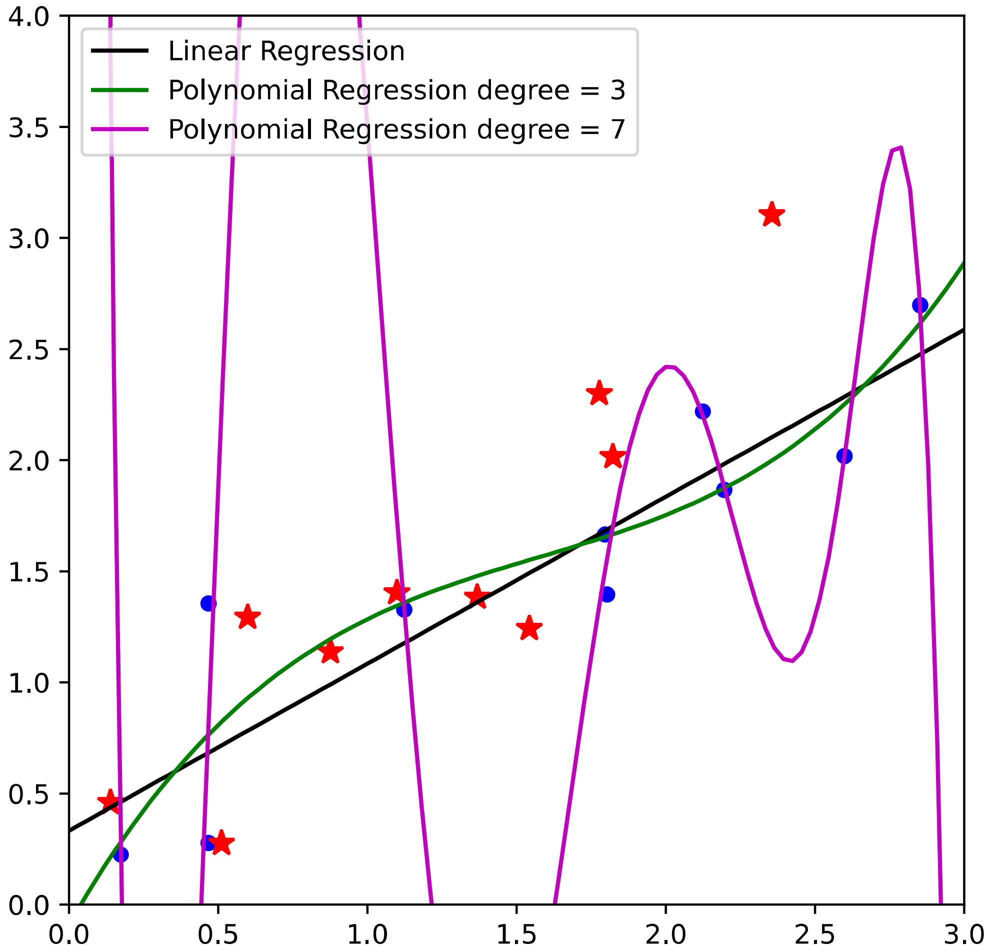

```
Training Loss:  0.06322418953679351
Testing Loss:  5.957556908635877
```

In [15]:

```
plt.figure(figsize=(6,6))
plt.plot(X_train, y_train,'b.', ms=10)
plt.plot(X_test, y_test,'r*', ms=10)
plt.plot(X_new, lin_reg.predict(X_new), 'k-', label='Linear Regression')
print('---Simple Linear Regression---')
print(lin_reg.intercept_, lin_reg.coef_)
lr_loss_train = mean_squared_error(y_train, lin_reg.predict(X_train))
print('Training Loss: ', lr_loss_train)
lr_loss = mean_squared_error(y_test, lin_reg.predict(X_test))
print('Testing Loss: ', lr_loss)
print()

deg = 3
print(f'---Polynomial Regression degree = {deg}---')
poly = PolynomialFeatures(degree=deg, include_bias=False)
X_poly = poly.fit_transform(X_train) # x --> x, x^2, x^3
X_new_poly = poly.fit_transform(X_new) # x --> x, x^2, x^3

poly_reg = LinearRegression()  # In essence, Polynomial Regression is still Line
poly_reg.fit(X_poly,y_train)
plt.plot(X_new, poly_reg.predict(X_new_poly), 'g', label='Polynomial Regression
print(poly_reg.intercept_, poly_reg.coef_)
lr_loss_train = mean_squared_error(y_train, poly_reg.predict(X_poly))
print('Training Loss: ', lr_loss_train)
X_test_poly = poly.fit_transform(X_test)
poly_3_loss = mean_squared_error(y_test, poly_reg.predict(X_test_poly))
print('Testing Loss: ', poly_3_loss)
print()
```

```python
deg = 7
print(f'---Polynomial Regression degree = {deg}---')
poly = PolynomialFeatures(degree=deg, include_bias=False)
X_poly = poly.fit_transform(X_train) # x --> x, x^2, x^3, ..., x^7
X_new_poly = poly.fit_transform(X_new) # x --> x, x^2, x^3, ... x^7

poly_reg = LinearRegression()
poly_reg.fit(X_poly, y_train)
plt.plot(X_new, poly_reg.predict(X_new_poly), 'm', label='Polynomial Regression
print(poly_reg.intercept_, poly_reg.coef_)
lr_loss_train = mean_squared_error(y_train, poly_reg.predict(X_poly))
print('Training Loss: ', lr_loss_train)
X_test_poly = poly.fit_transform(X_test)
poly_7_loss = mean_squared_error(y_test, poly_reg.predict(X_test_poly))
print('Testing Loss: ', poly_7_loss)

plt.axis([0, 3, 0, 4])
plt.legend()
plt.show()
```

```
---Simple Linear Regression---
[0.33181066] [[0.75155622]]
Training Loss:  0.09991531905633781
Testing Loss:  0.21020128681013045

---Polynomial Regression degree = 3---
[-0.0916451] [[ 2.35120734 -1.23811156  0.26177074]]
Training Loss:  0.0875019006667126
Testing Loss:  0.23747048298743464

---Polynomial Regression degree = 7---
[38.069871] [[ -399.60873887  1364.69579468 -2093.21939506  1665.82047777
   -716.24860915   158.03695709   -14.0390417 ]]
Training Loss:  0.06322418953679351
Testing Loss:  5.957556908635877
```



The 7-th degree polynomial regression model above (magenta color) is overfit to our training dataset. If we use an overfit model on our future data, it will have a lot of errors.

The magnitude of parameters of an overfit model would be very big (very far from 0.0).

Ridge Regression

Ridge Regression: Weight Decay/Maximum A Posteriori Estimation (MAP) for Linear Regression

$\texttt{prediction} = w_0 + w_1 x + w_2 x^2 + \ldots + w_D x^D$ (However $w_1, w_2, \ldots, w_D$ will have small values close to 0.0).

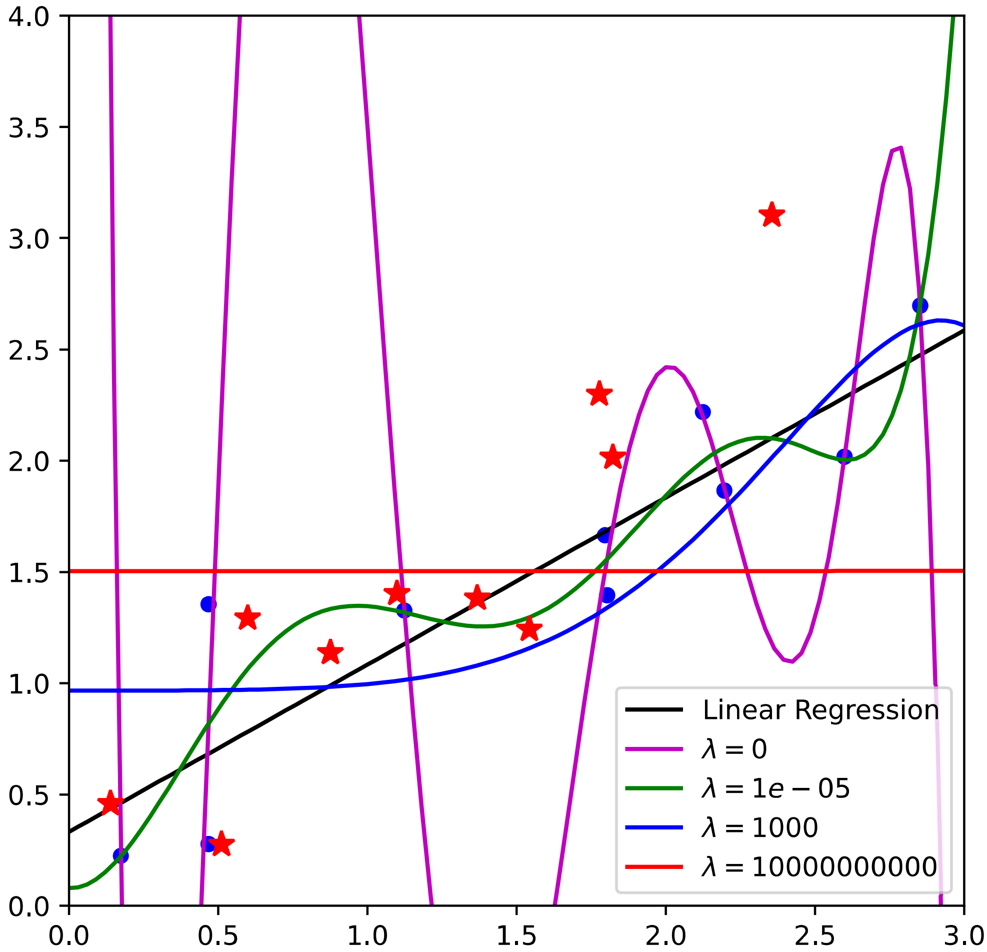
$w_{map} = \arg\min_w NLL(w) + \lambda ||w||_2^2$

$\lambda ||w||_2^2$: Regularziation term, and λ is the regularization strength.

In [16]:
```
plt.figure(figsize=(6,6))
plt.plot(X_train, y_train,'b.', ms=10)
plt.axis([0, 3, 0, 4])

deg = 7
poly = PolynomialFeatures(degree=deg, include_bias=False)
X_poly = poly.fit_transform(X_train) # x ---> x, x^2, x^3, ..., x^7
X_new_poly = poly.fit_transform(X_new)
```

```
plt.plot(X_new, lin_reg.predict(X_new), 'k-', label='Linear Regression')
print(lin_reg.intercept_, lin_reg.coef_)

lambd = 0
ridge_reg = Ridge(alpha=lambd)
ridge_reg.fit(X_poly, y_train)
plt.plot(X_new, ridge_reg.predict(X_new_poly), 'm-', label=r'$\lambda={}$'.forma
print(ridge_reg.intercept_, ridge_reg.coef_)

lambd = 10**-5  # 10^{-5}
ridge_reg = Ridge(alpha=lambd)
ridge_reg.fit(X_poly, y_train)
plt.plot(X_new, ridge_reg.predict(X_new_poly), 'g-', label=r'$\lambda={}$'.forma
print(ridge_reg.intercept_, ridge_reg.coef_)

lambd = 1000
ridge_reg = Ridge(alpha=lambd)
ridge_reg.fit(X_poly, y_train)
plt.plot(X_new, ridge_reg.predict(X_new_poly), 'b-', label=r'$\lambda={}$'.forma
print(ridge_reg.intercept_, ridge_reg.coef_)

lambd = 10000000000  # Choose it by our experience, trial and error
ridge_reg = Ridge(alpha=lambd)
ridge_reg.fit(X_poly, y_train)
plt.plot(X_new, ridge_reg.predict(X_new_poly), 'r-', label=r'$\lambda={}$'.forma
print(ridge_reg.intercept_, ridge_reg.coef_)

plt.plot(X_test, y_test,'r*', ms=10)
plt.legend()
plt.show()
```

```
[0.33181066] [[0.75155622]]
[38.06834626] [[ -399.59270239  1364.64125612 -2093.13554384  1665.75343429
   -716.21961323   158.03051773   -14.03846577]]
[0.07987999] [[-0.09370931  6.14945012 -4.77478402 -3.78021829  5.75140457 -2.283
94658
   0.29814782]]
[0.96643294] [[ 0.00141423  0.00266913  0.00442294  0.00683627  0.00931761  0.008
96911
  -0.00359498]]
[1.50446301] [[6.07278017e-10 1.77351966e-09 4.72471362e-09 1.25448975e-08
  3.35844395e-08 9.07463757e-08 2.47260904e-07]]
```


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Linear Regression
$\lambda = 0$
$\lambda = 1e-05$
$\lambda = 1000$
$\lambda = 10000000000$

In [16]: