# Lab 2 by Anthony Bilic 20514128

# Problem 1: Linear Regression (60 points)

## 1.1

In [1]:
```
from __future__ import division
import numpy as np
import mltools as ml
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(0)
```

In [2]:
```
data = np.genfromtxt("data/curve80.txt", delimiter=None) # Loading the text data file
X = data[:,0] #scalar feature value x;
X = np.atleast_2d(X).T # code expects shape (M,N) so make sure it's 2-dimensional
Y = data[:,1] #target value y for each example.
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75) # split data set 75/25
#do not reorder (shuffle) the data
```

**Print the shapes of these four objects. (5 points)**

In [3]:
```
Xtr.shape
```

Out[3]: (60, 1)

In [4]:
```
Xte.shape
```

Out[4]: (20, 1)

In [5]:
```
Ytr.shape
```

Out[5]: (60,)

In [6]:
```
Yte.shape
```

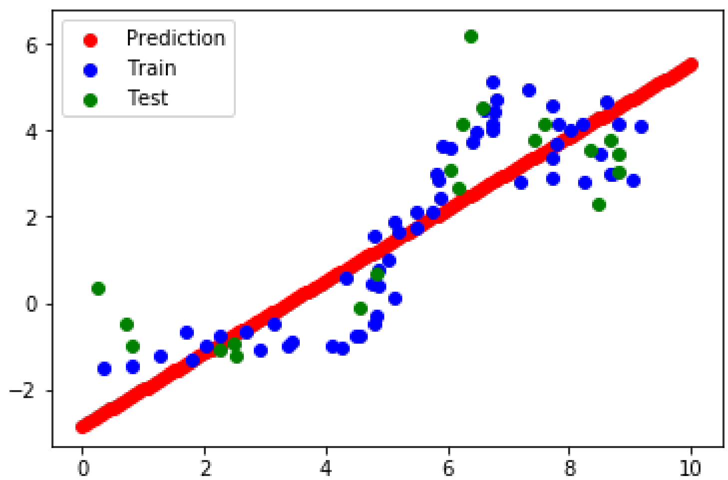
Out[6]: (20,)

## 1.2

In [7]:
```
lr = ml.linear.linearRegress( Xtr, Ytr ) # create and train model
xs = np.linspace(0,10,200) # densely sample possible x-values
xs = xs[:,np.newaxis] # force "xs" to be an Mx1 matrix
ys = lr.predict( xs ) # make predictions at xs
```

**(a) Plot the training data points along with your prediction function in a single plot. (10 points)**

In [8]:
```
f, ax = plt.subplots()
ax.scatter(xs, ys, color='red', label='Prediction')
ax.scatter(Xtr, Ytr, color='blue', label='Train')
ax.scatter(Xte, Yte, color='green', label='Test')
plt.legend()
plt.show()
```




**(b) Print the linear regression coefficients ( lr.theta ) and verify that they match your plot. (5 points)**

In [9]:
```
print(lr.theta)
```

```
[[-2.82765049  0.83606916]]
```

**(c) What is the mean squared error of the predictions on the training and test data? (10 points)**

In [10]:
```
def MSE(y_true, y_hat):
    n = len(y_true)
    total = 0
    for i in range(n):
        total += (y_true[i,] - y_hat[i,])
    totalSquared = total**2
    return totalSquared / n
```

In [11]:
```
print("Training data")
print(MSE(Ytr, ys))
```

```
Training data
[ 708.24922863]
```

In [12]:
```
print("Test data")
print(MSE(Yte, ys))
```

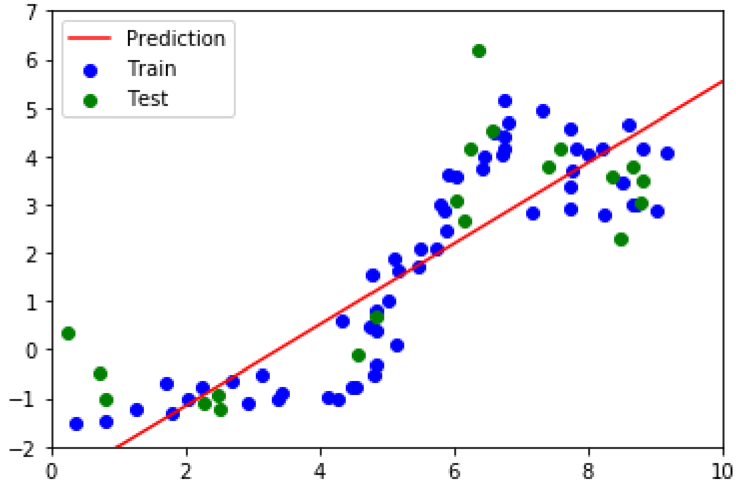
```
Test data
[ 401.09080577]
```

## 1.3

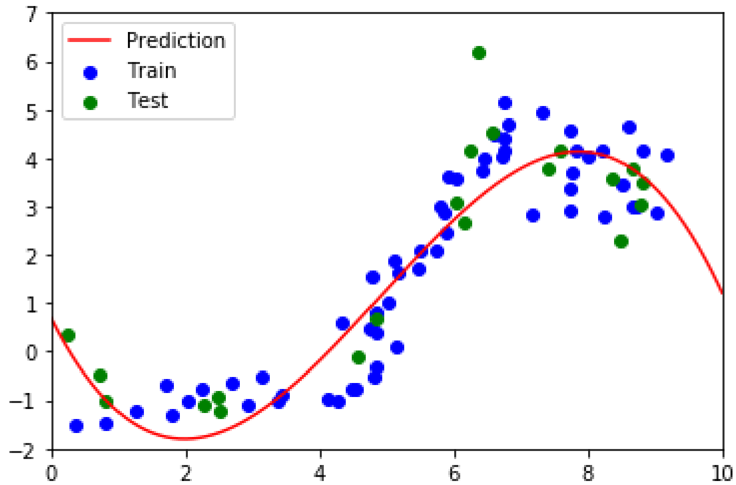
### (a) For each model, plot the learned prediction function f (x). (15 points)

In [13]:
```
degrees = np.array([1, 3, 5, 7, 10, 18])
mse_error = np.zeros(degrees.shape[0])
mse_error2 = np.zeros(degrees.shape[0])

for i, degree in enumerate(degrees):
    # Create polynomial features up to "degree"; don't create constant f
eature
    # (the linear regression learner will add the constant feature autom
atically)
    XtrP = ml.transforms.fpoly(Xtr, degree, bias=False)

    # Rescale the data matrix so that the features have similar ranges /
 variance
    XtrP,params = ml.transforms.rescale(XtrP)
    # "params" returns the transformation parameters (shift & scale)

    # Then we can train the model on the scaled feature matrix:
    lr = ml.linear.linearRegress( XtrP, Ytr ) # create and train model

    # Now, apply the same polynomial expansion & scaling transformation
 to Xtest:
    XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte, degree, bia
s=False), params)

    #=================================================================
=================

    xs = np.linspace(0, 10, 200)
    xs = np.atleast_2d(xs).T
    xsP,_ = ml.transforms.rescale(ml.transforms.fpoly(xs, degree, bias=F
alse), params)
    ys = lr.predict(xsP)

    f, ax = plt.subplots()
    ax.plot(xs, ys, color='red', label='Prediction')
    ax.scatter(Xtr, Ytr, color='blue', label='Train')
    ax.scatter(Xte, Yte, color='green', label='Test')
    ax.set_ylim(-2, 7) # Set the minimum and maximum limits
    ax.set_xlim(0, 10) # Set the minimum and maximum limits

    print(degree)
    plt.legend()
    plt.show()

    mse_error[i] = MSE(Ytr, ys)
    mse_error2[i] = MSE(Yte, ys)
```

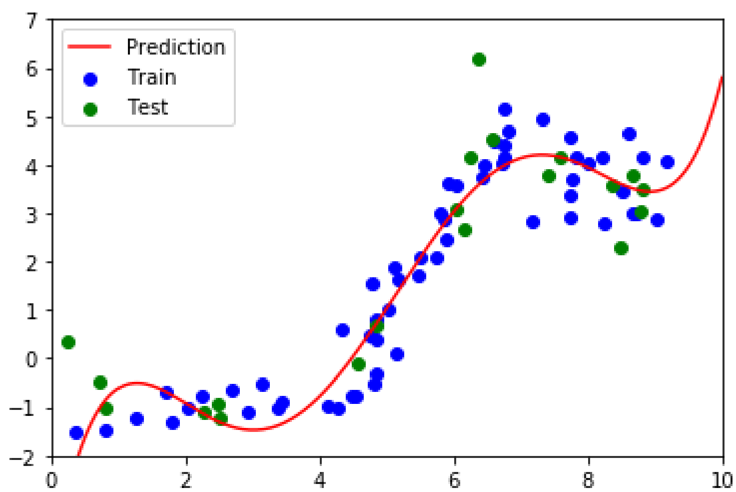
1


Prediction
Train
Test

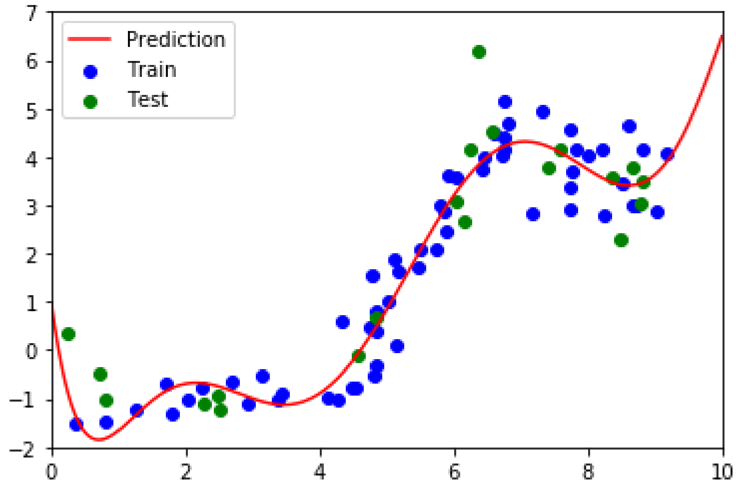
3


Prediction
Train
Test

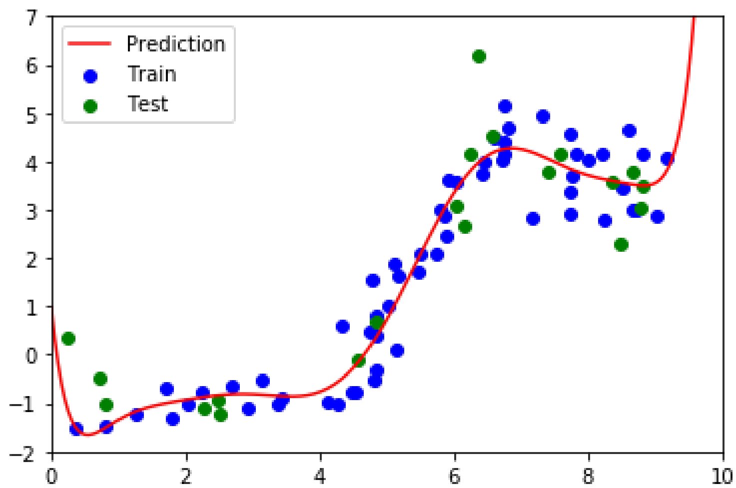
5


Prediction
Train
Test

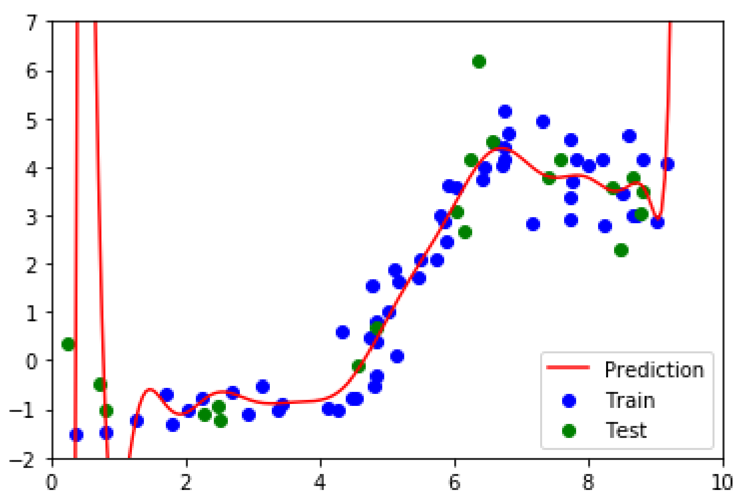
7




10



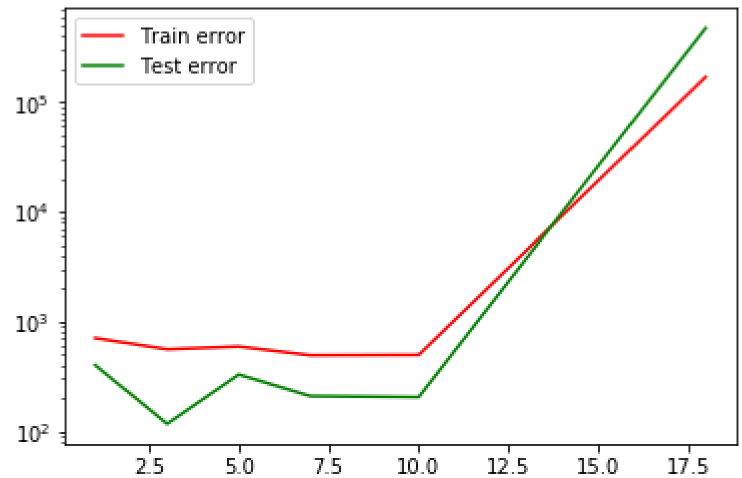

18




**(b) Plot the training and test errors on a log scale ( semilogy ) as a function of the model degree. (10 points)**

In [14]:
```
f, ax = plt.subplots()

# Plotting a line with markers where there's an actual x value.
print("MSE ERROR")
ax.semilogy(degrees, mse_error, color='red', label="Train error")
ax.semilogy(degrees, mse_error2, color='green', label="Test error")
plt.legend()
plt.show()
```

MSE ERROR



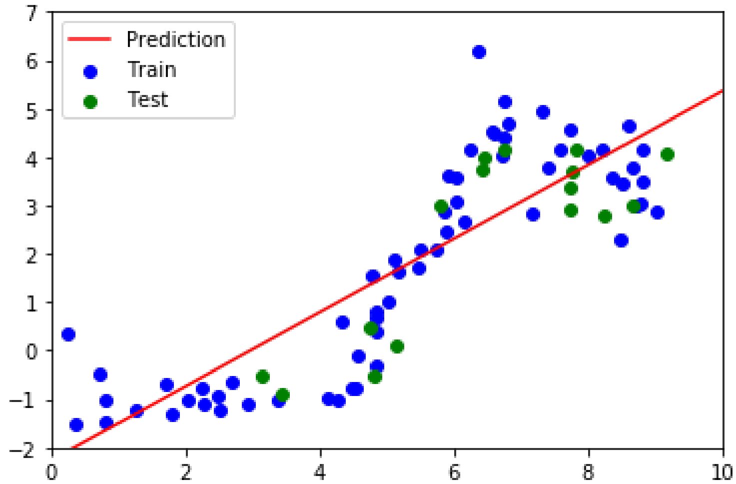

**(c) What polynomial degree do you recommend? (5 points)**

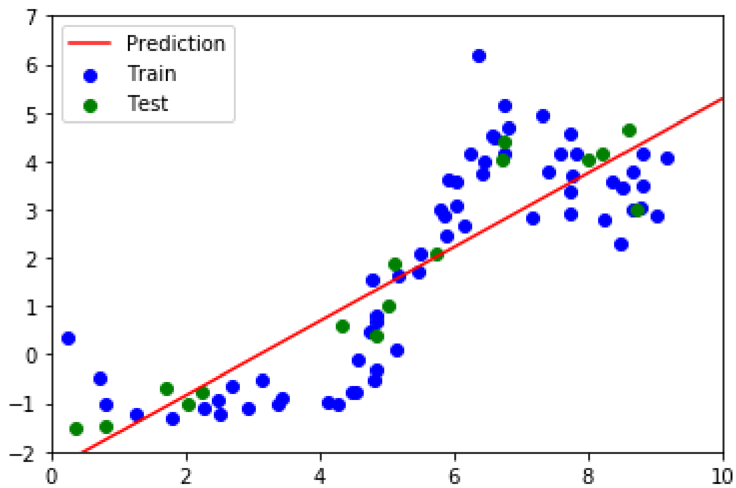
I would recommend around degree 3 as it has the least total error.

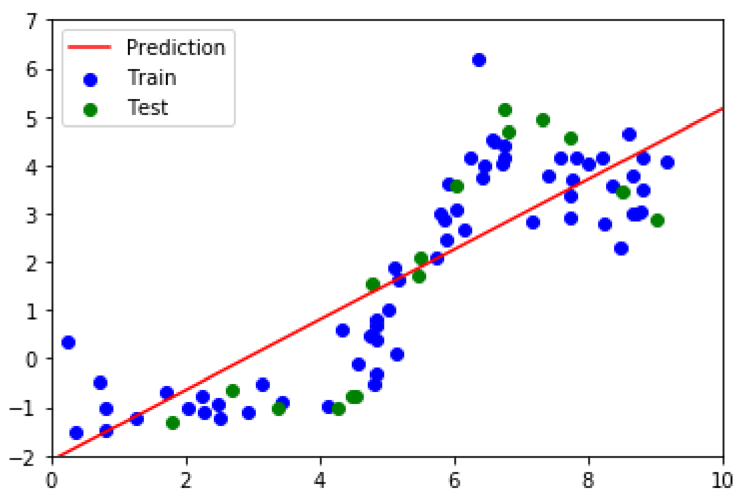
# Problem 2: Cross-validation (35 points)

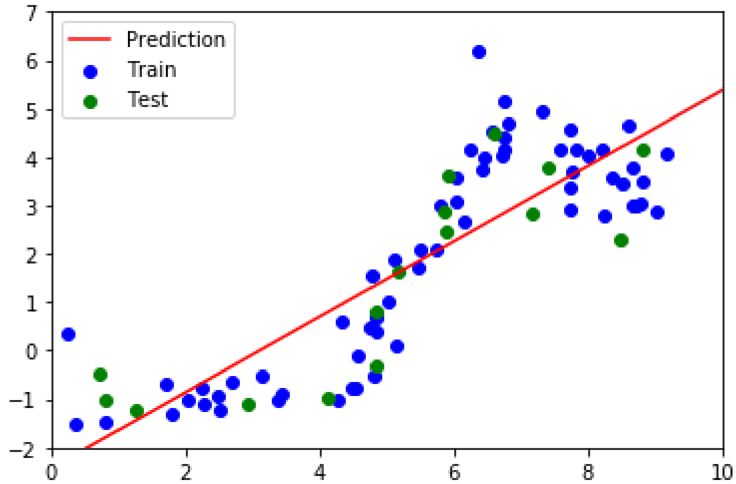
## 2.1 Plot the five-fold cross-validation error (with semilogy , as before) as a function of degree. (10 points)

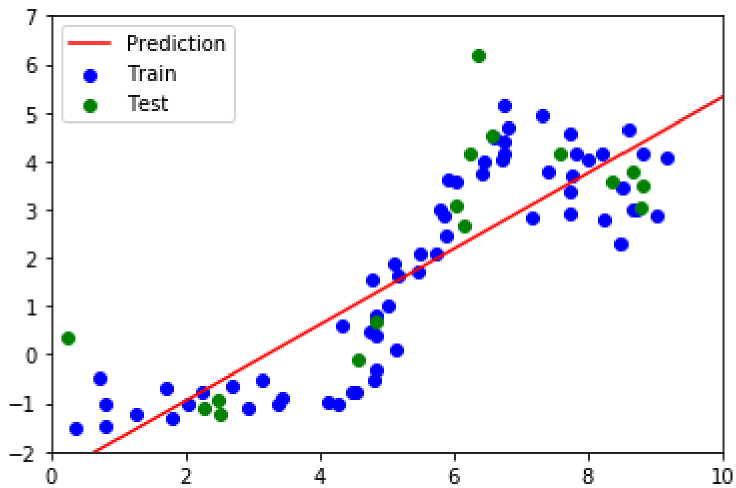
In [15]:
```
nFolds = 5
J = np.zeros(nFolds)
J2 = np.zeros(nFolds)

degrees = np.array([1, 3, 5, 7, 10, 18])
mse_error = np.zeros(degrees.shape[0])
mse_error2 = np.zeros(degrees.shape[0])

for i, degree in enumerate(degrees):
    for iFold in range(nFolds):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(X,Y,nFolds,iFold) # use ith b
lock as validation
        learner = ml.linear.linearRegress(Xti,Yti)

        XtiP = ml.transforms.fpoly(Xti, degree, bias=False)
        XtiP,params = ml.transforms.rescale(XtiP)
        learner = ml.linear.linearRegress( XtiP, Yti ) # create and trai
n model
        XviP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xvi, degree,
 bias=False), params)

        xs = np.linspace(0, 10, 200)
        xs = np.atleast_2d(xs).T
        xsP,_ = ml.transforms.rescale(ml.transforms.fpoly(xs, degree, bi
as=False), params)
        ys = learner.predict(xsP)

        f, ax = plt.subplots()
        ax.plot(xs, ys, color='red', label='Prediction')
        ax.scatter(Xti, Yti, color='blue', label='Train')
        ax.scatter(Xvi, Yvi, color='green', label='Test')

        ax.set_ylim(-2, 7) # Set the minimum and maximum limits
        ax.set_xlim(0, 10) # Set the minimum and maximum limits
        plt.legend()
        plt.show()

        J[iFold] = MSE(Yti, ys)
        J2[iFold] = MSE(Yvi, ys)

    mse_error[i] = np.mean(J)
    mse_error2[i] = np.mean(J2)
```

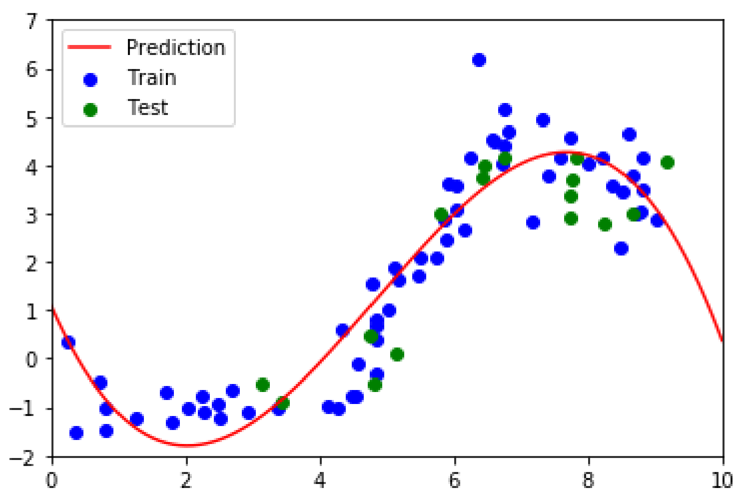

Prediction
Train
Test

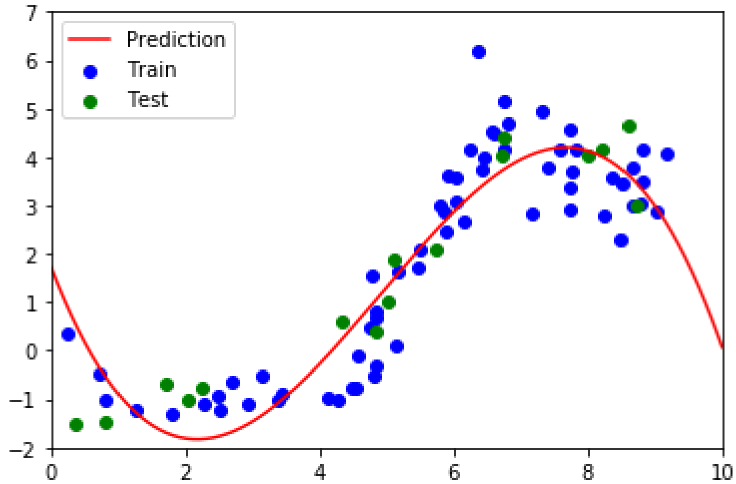

Prediction
Train
Test

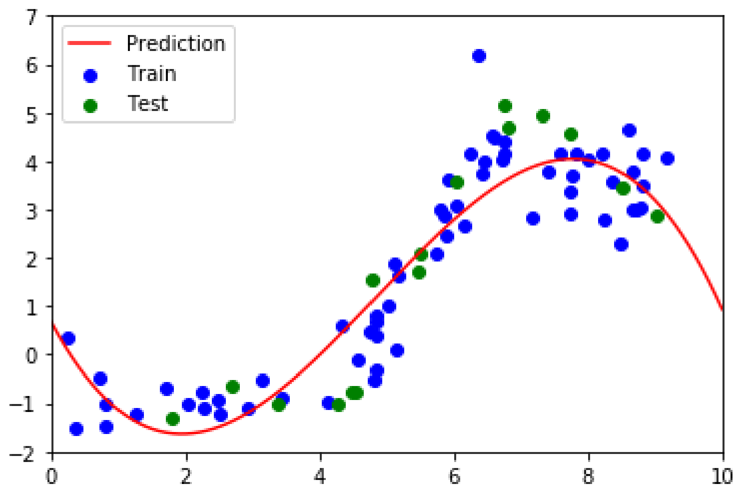

Prediction
Train
Test

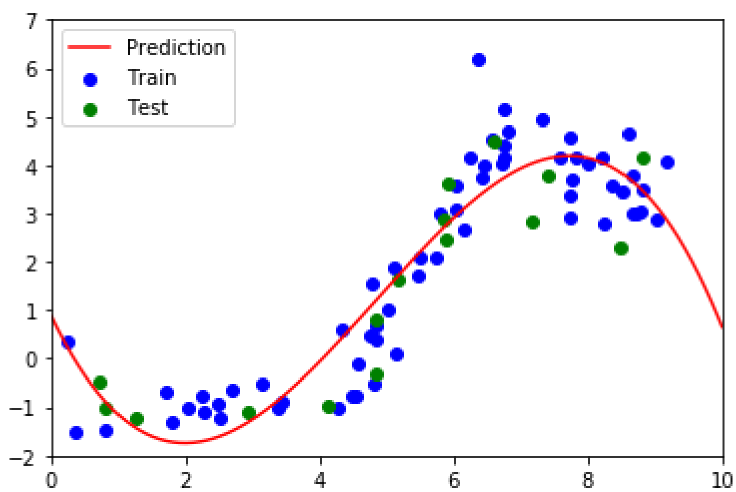

Prediction
Train
Test
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10

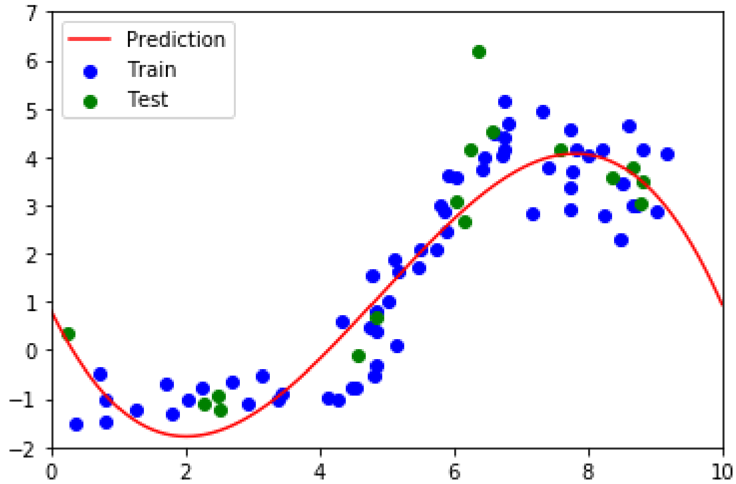

Prediction
Train
Test
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10

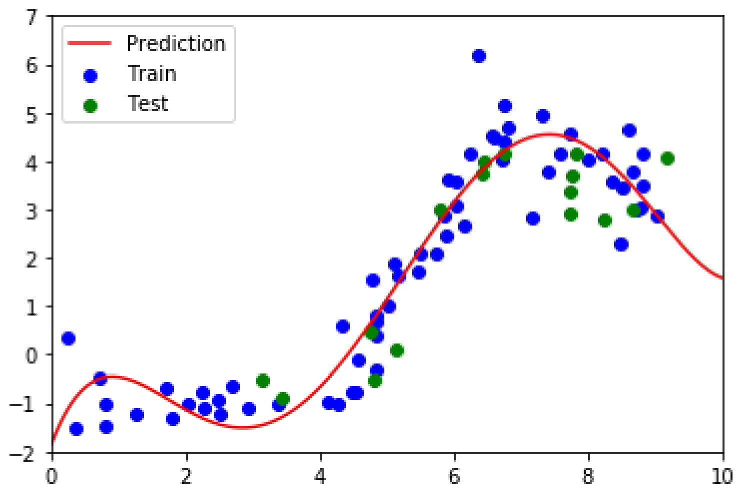

Prediction
Train
Test
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10

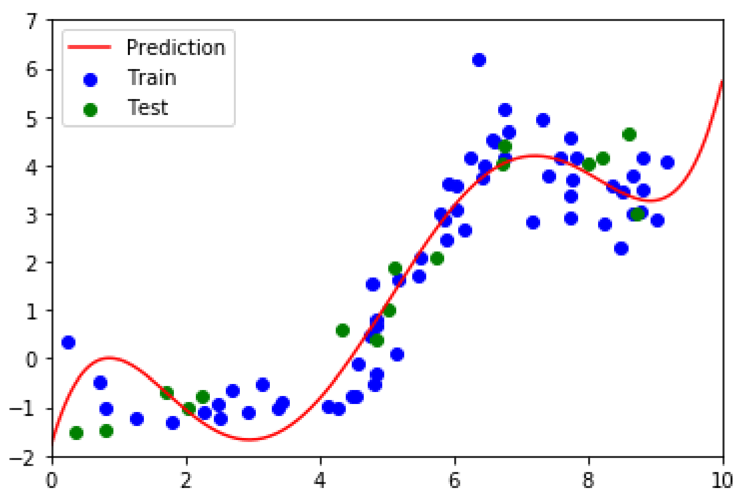

Prediction
Train
Test
7 6 5 4 3 2 1 0 −1 −2
0 2 4 6 8 10

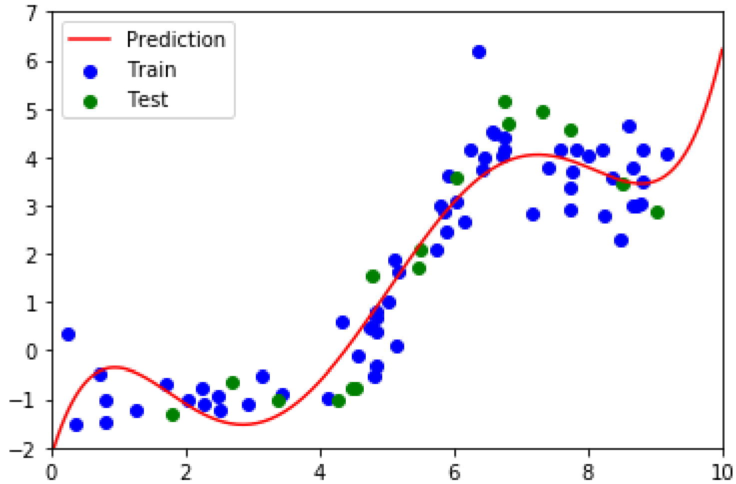

Prediction
Train
Test
7 6 5 4 3 2 1 0 −1 −2
0 2 4 6 8 10

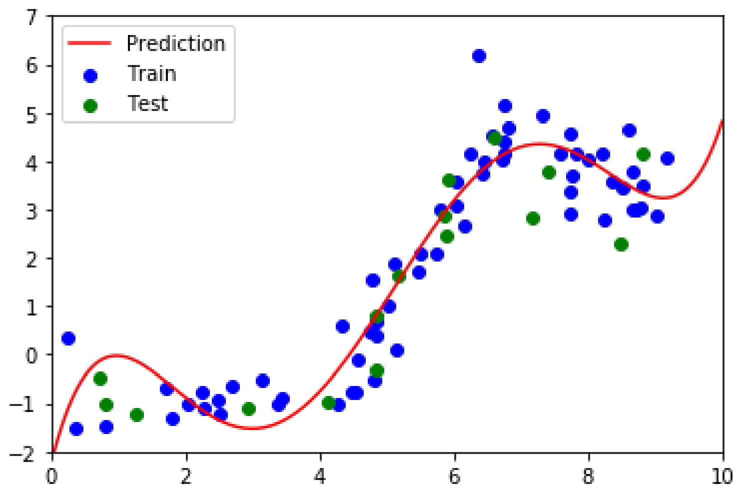

Prediction
Train
Test
7 6 5 4 3 2 1 0 −1 −2
0 2 4 6 8 10

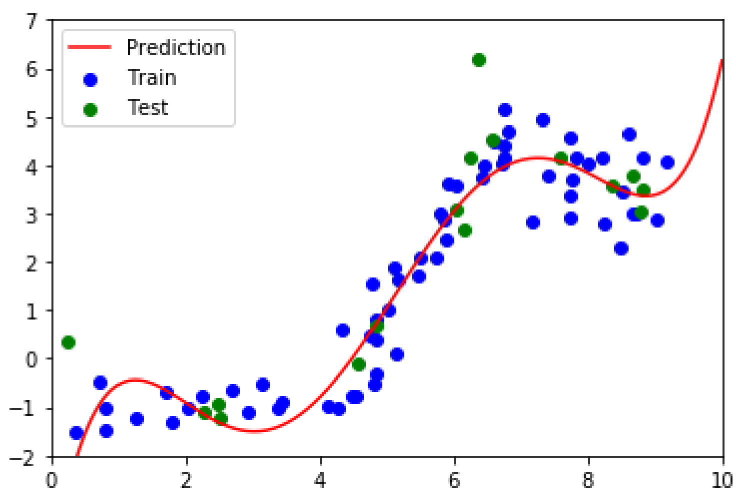

Prediction
Train
Test

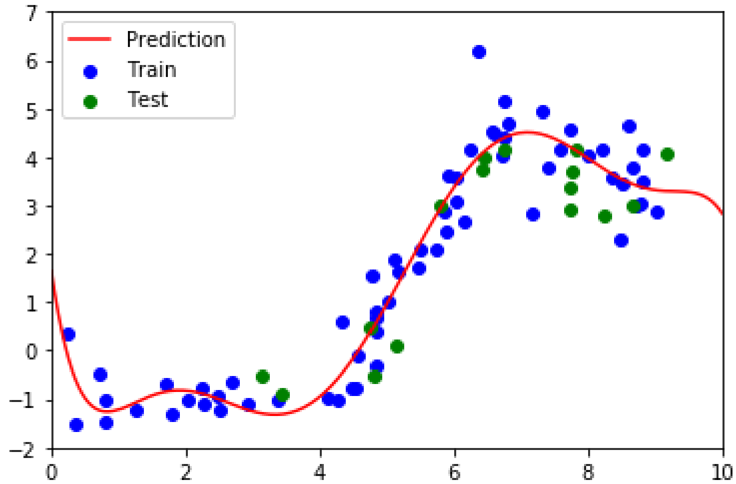

Prediction
Train
Test

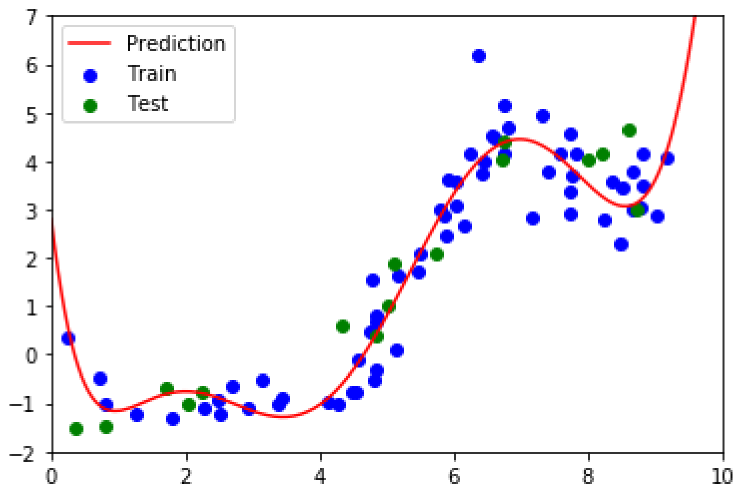

Prediction
Train
Test

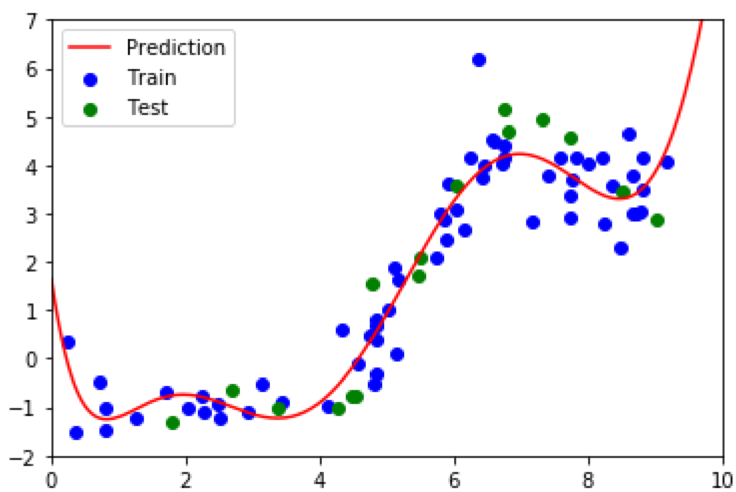

Prediction
Train
Test
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10

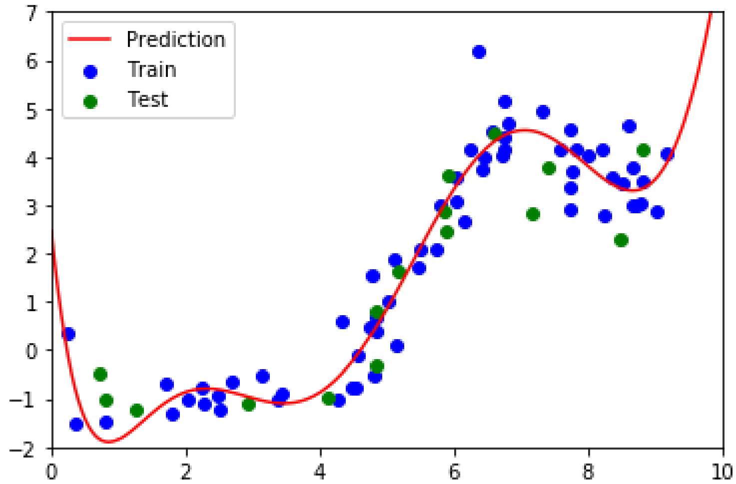

Prediction
Train
Test
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10

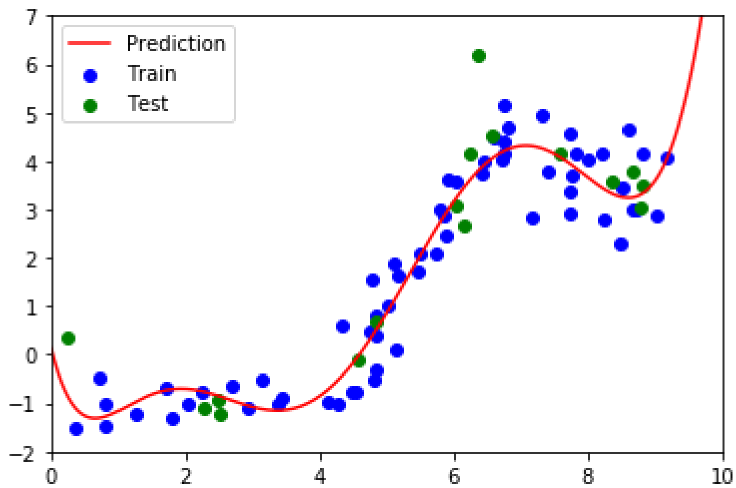

Prediction
Train
Test
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
8
10

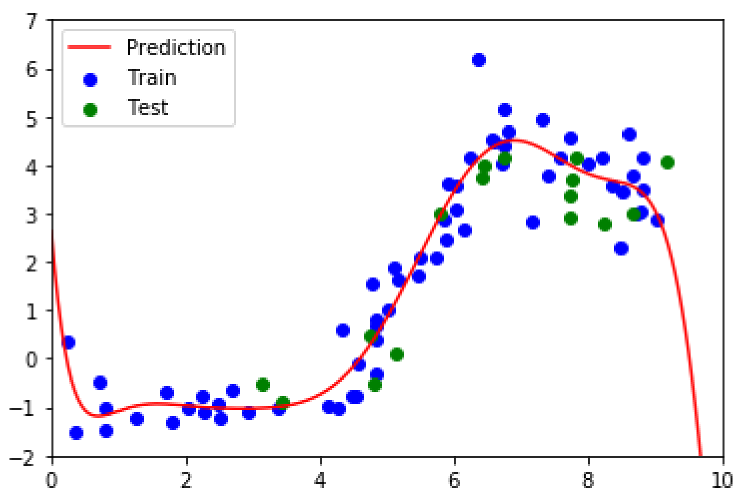

Prediction
Train
Test

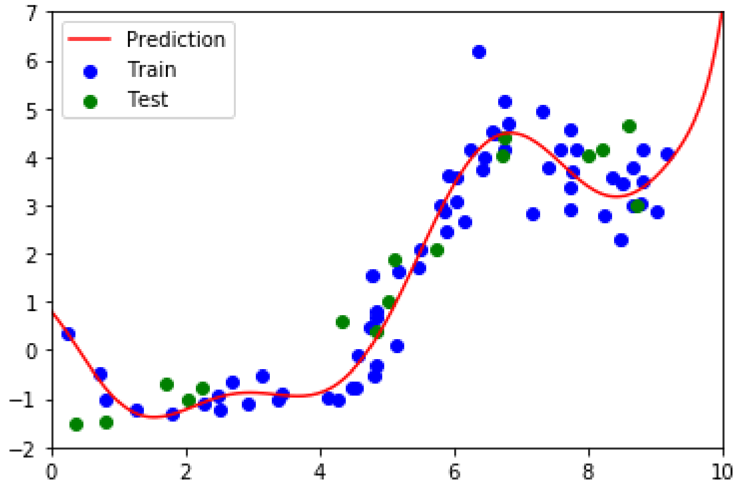

Prediction
Train
Test

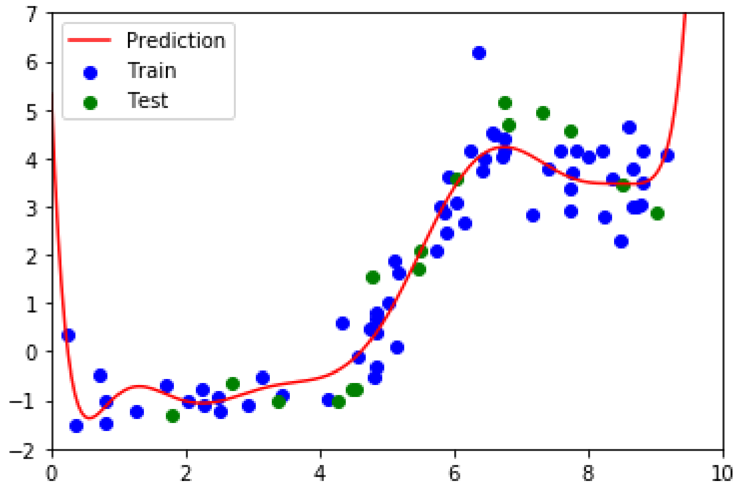

Prediction
Train
Test

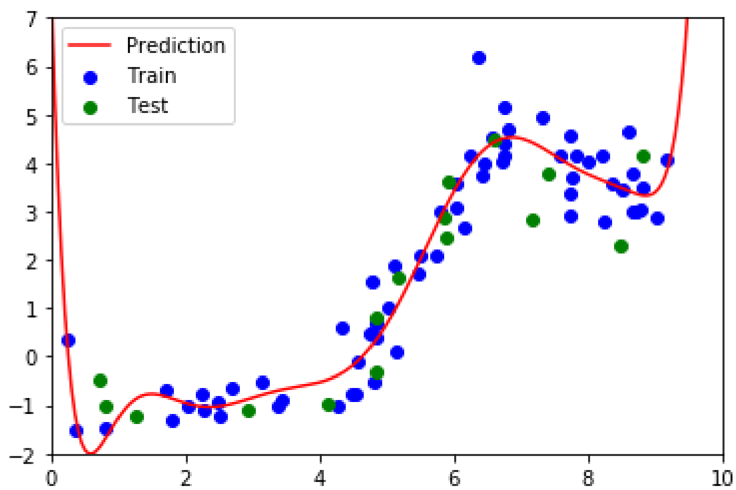

Prediction
Train
Test

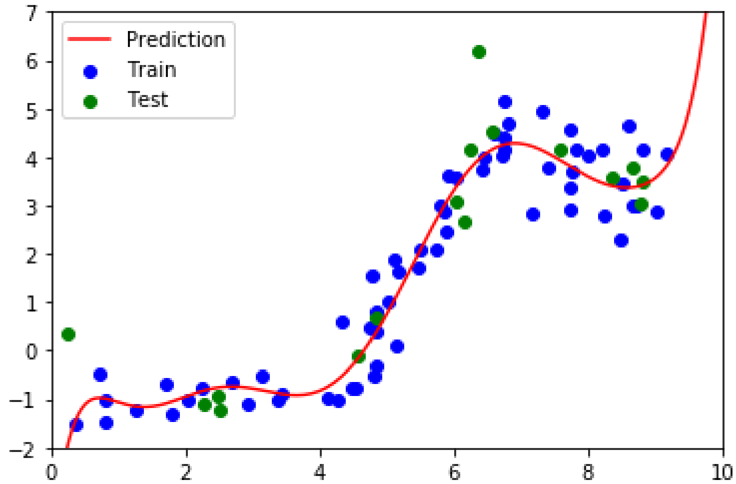

Prediction
Train
Test

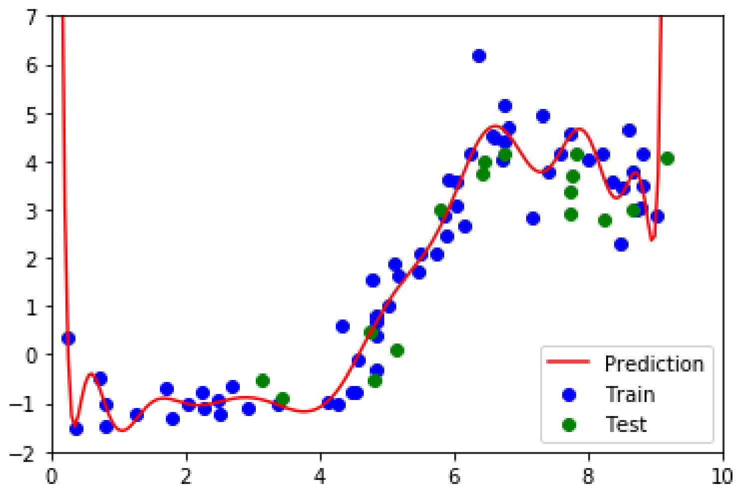

Prediction
Train
Test

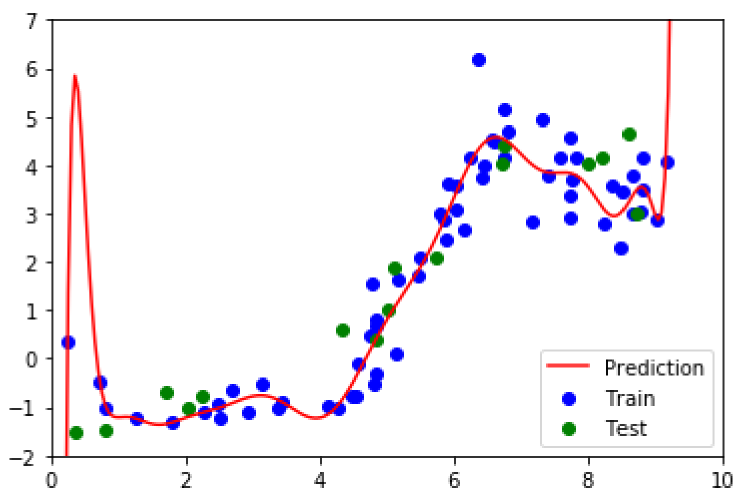

Prediction
Train
Test

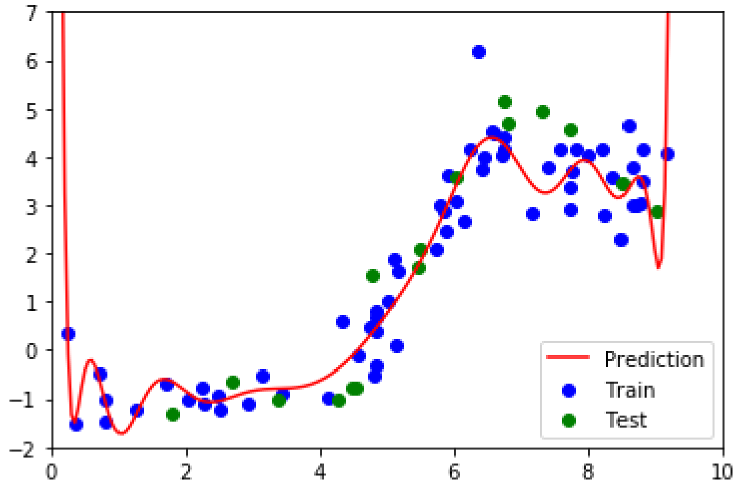

Prediction
Train
Test

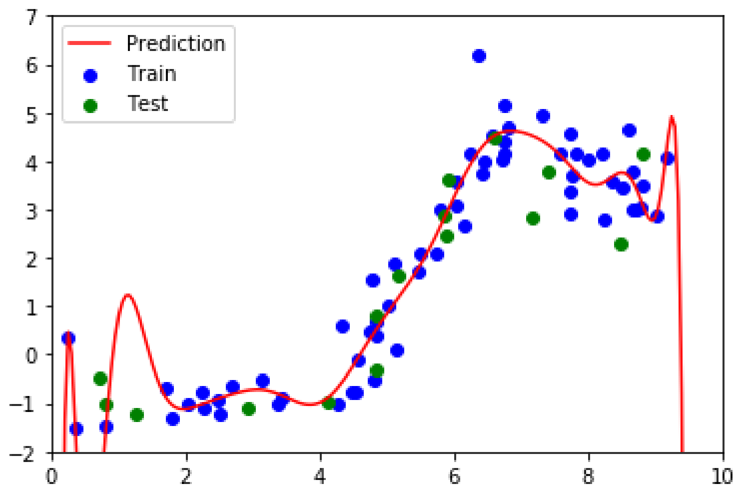

Prediction
Train
Test

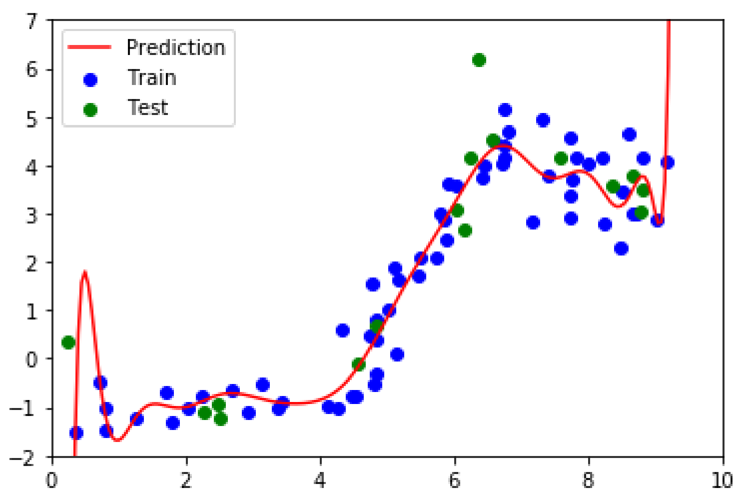

Prediction
Train
Test


Prediction
Train
Test


Prediction
Train
Test

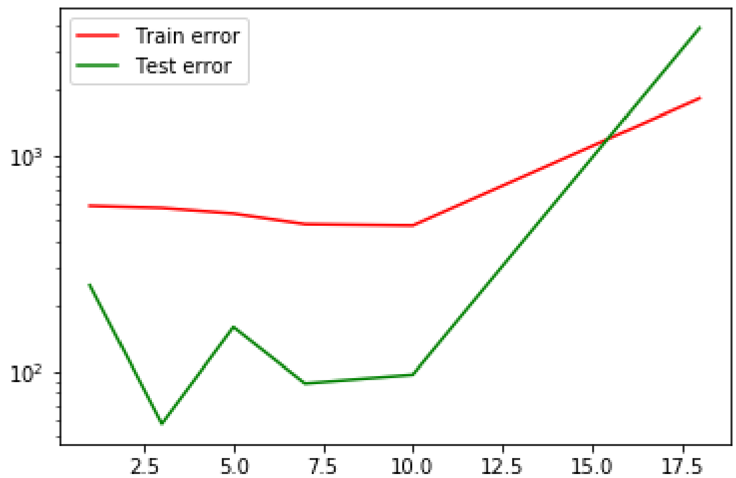

Prediction
Train
Test


Prediction
Train
Test


Prediction
Train
Test

In [16]:
```
f, ax = plt.subplots()
print("MSE ERROR")

ax.semilogy(degrees, mse_error, color='red', label="Train error")
ax.semilogy(degrees, mse_error2, color='green', label="Test error")
plt.legend()
plt.show()
```

MSE ERROR




### 2.2 How do the MSE estimates from five-fold cross-validation compare to the MSEs evaluated on the actual test data (Problem 1)? (5 points)

The cross-validation MSEs are lower. Thus it has less error and is more effective.

### 2.3 Which polynomial degree do you recommend based on five-fold cross-validation error? (5 points)

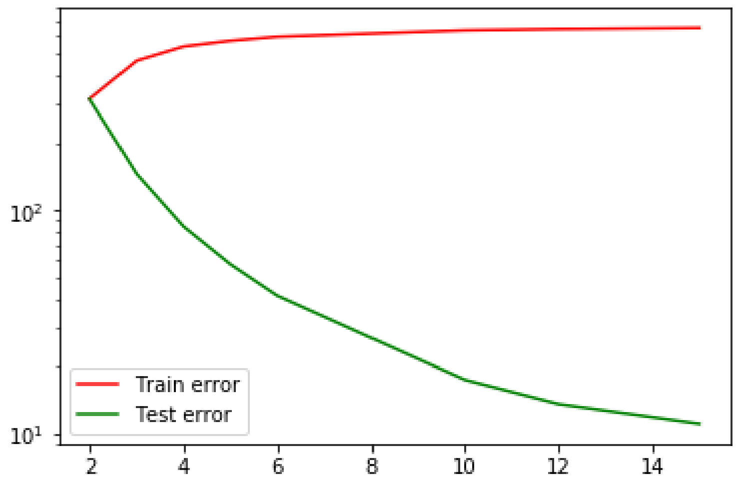
I would recommend around degree 3 as it has the least total error.

### 2.4 For the degree that you picked in step 3, plot (with semilogy ) the cross-validation error as the number of folds is varied from nFolds = 2, 3, 4, 5, 6, 10, 12, 15. What pattern do you observe, and how do you explain why it occurs? (15 points)

In [20]:
```
folds = np.array([2, 3, 4, 5, 6, 10, 12, 15])
mse_error = np.zeros(folds.shape[0])
mse_error2 = np.zeros(folds.shape[0])
chosenDegree = 3

for i, nFolds in enumerate(folds):
    J = np.zeros(nFolds)
    J2 = np.zeros(nFolds)

    for iFold in range(nFolds):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(X,Y,nFolds,iFold) # use ith b
lock as validation
        learner = ml.linear.linearRegress(Xti,Yti)

        XtiP = ml.transforms.fpoly(Xti, chosenDegree, bias=False)
        XtiP,params = ml.transforms.rescale(XtiP)
        learner = ml.linear.linearRegress( XtiP, Yti ) # create and trai
n model
        XviP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xvi, chosenD
egree, bias=False), params)

        xs = np.linspace(0, 10, 200)
        xs = np.atleast_2d(xs).T
        xsP,_ = ml.transforms.rescale(ml.transforms.fpoly(xs, chosenDegr
ee, bias=False), params)
        ys = learner.predict(xsP)

        J[iFold] = MSE(Yti, ys)
        J2[iFold] = MSE(Yvi, ys)

    mse_error[i] = np.mean(J)
    mse_error2[i] = np.mean(J2)
```

In [21]:
```
f, ax = plt.subplots()
print("MSE ERROR")

ax.semilogy(folds, mse_error, color='red', label="Train error")
ax.semilogy(folds, mse_error2, color='green', label="Test error")
plt.legend()
plt.show()
```

MSE ERROR




Train error grows very slowly while testing error shrink drastically. Thus the more cross-validation the more fit our model becomes to the validation data.

## Problem 3: Statement of Collaboration (5 points)

I did it all by myself.