HOMEWORK 2 CS 178

PROBLEM 1 : LINEAR REGRESSION

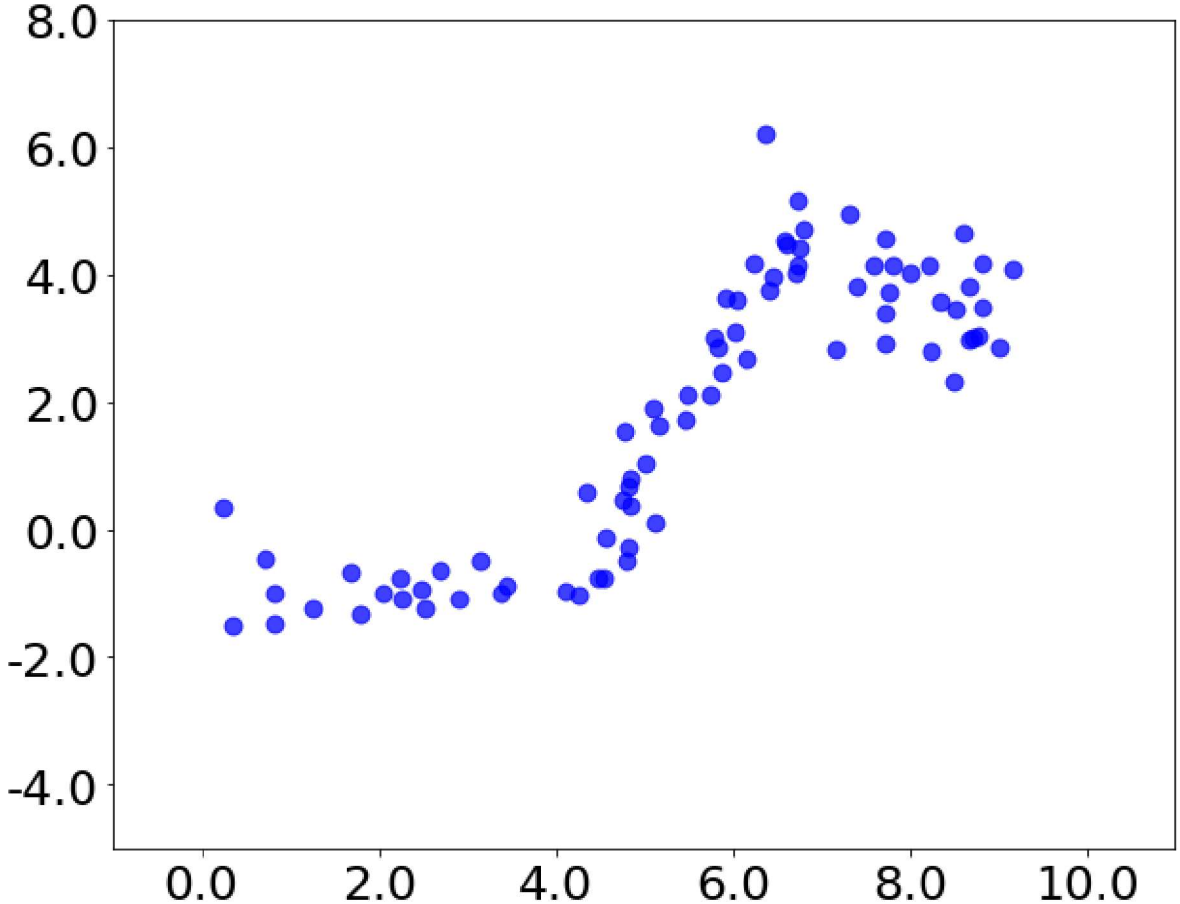
In [286]:
```
from __future__ import division

import numpy as np
import matplotlib.pyplot as plt
import mltools as ml

np.random.seed(0)
%matplotlib inline


## load data

data = np.genfromtxt("C:/Users/Sergio/Desktop/Fall 2017/CS178- Data Mining/HW2
-code/data/curve80.txt" ,delimiter = None)


X = data[:,0]
X = np.atleast_2d(X).T # code expects shape (M,N) so make sure it's 2-dimensio
nal
Y = data[:,1] # doesn't matter for Y
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75)


print('X.shape: ', X.shape)
print('Y.shape: ', Y.shape)
print('Xtr.shape: ', Xtr.shape)
print('Xte.shape: ', Xte.shape)
print('Ytr.shape: ', Ytr.shape)
print('Yte.shape: ', Yte.shape)
print('\n\n')

# Xtr has .75 the the first 60 Xte has .25 the last 20 (same with Ytr/Yte)


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 (expected by our code)
ys = lr.predict( xs ) # make predictions at xs




# This prints part C
print( "Linear Regression Coefficient: ", lr.theta)
print(lr) # this also gets the linear reg coeff.
```

```
# in obtaining the mse it was not as clear from discussion notes
# I was browsing over the mltools and notice
# we have a function that computes the MSE in the base file.

''' Computes the mean squared error

    Computes
      (1/M) \sum_i ( f(x^{(i)}) - y^{(i)} )^2
    of a regression model f(.) on test data X and Y.

    Args:
      X (arr): M x N array that contains M data points with N features
      Y (arr): M x 1 array of target values for each data point

    Returns:
      float: mean squared error

    BEHIND THE SCENES IT IS DOING

    Yhat = self.predict(X)
    return np.mean( (Y - Yhat.reshape(Y.shape))**2 , axis=0)
'''
# Here I print the MSE for training data and test data
print('MSE training data: ', lr.mse(Xtr, Ytr))
print('MSE test data: ', lr.mse(Xte, Yte))

# This next code snippet comes from discussion
# I will using it with with the data above


# Plotting the data
f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.scatter(X, Y, s=80, color='blue', alpha=0.75)

ax.set_xlim(-1, 11)
ax.set_ylim(-5, 8)
ax.set_xticklabels(ax.get_xticks(), fontsize=25)
ax.set_yticklabels(ax.get_yticks(), fontsize=25)

plt.show()







# We start with creating a set of xs on the space we want to predict for.
xs = np.linspace(0, 10, 200)

# Converting to the rate shape
xs = np.atleast_2d(xs).T

# And now the prediction
ys = lr.predict(xs)

#plt.rcParams['figure.figsize'] = (5.0, 3.0)
```

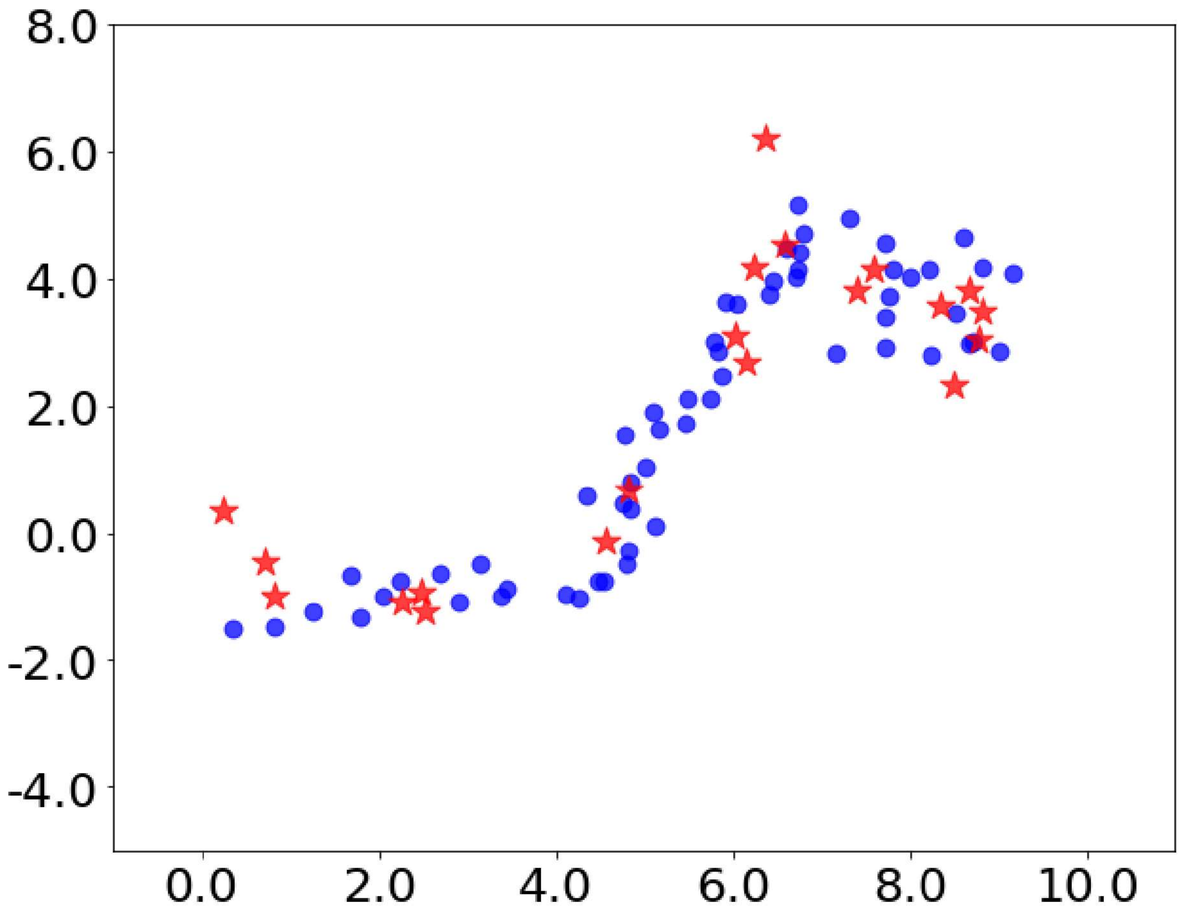
```
#lines = plt.plot(xs,ys,'k-',Xtr,Ytr,'r.',Xte,Yte,'g.', linewidth=3,markersize
=12)
#plt.legend(['Prediction','Train','Test'],loc='lower right');
```

```
X.shape:  (80, 1)
Y.shape:  (80,)
Xtr.shape:  (60, 1)
Xte.shape:  (20, 1)
Ytr.shape:  (60,)
Yte.shape:  (20,)
```

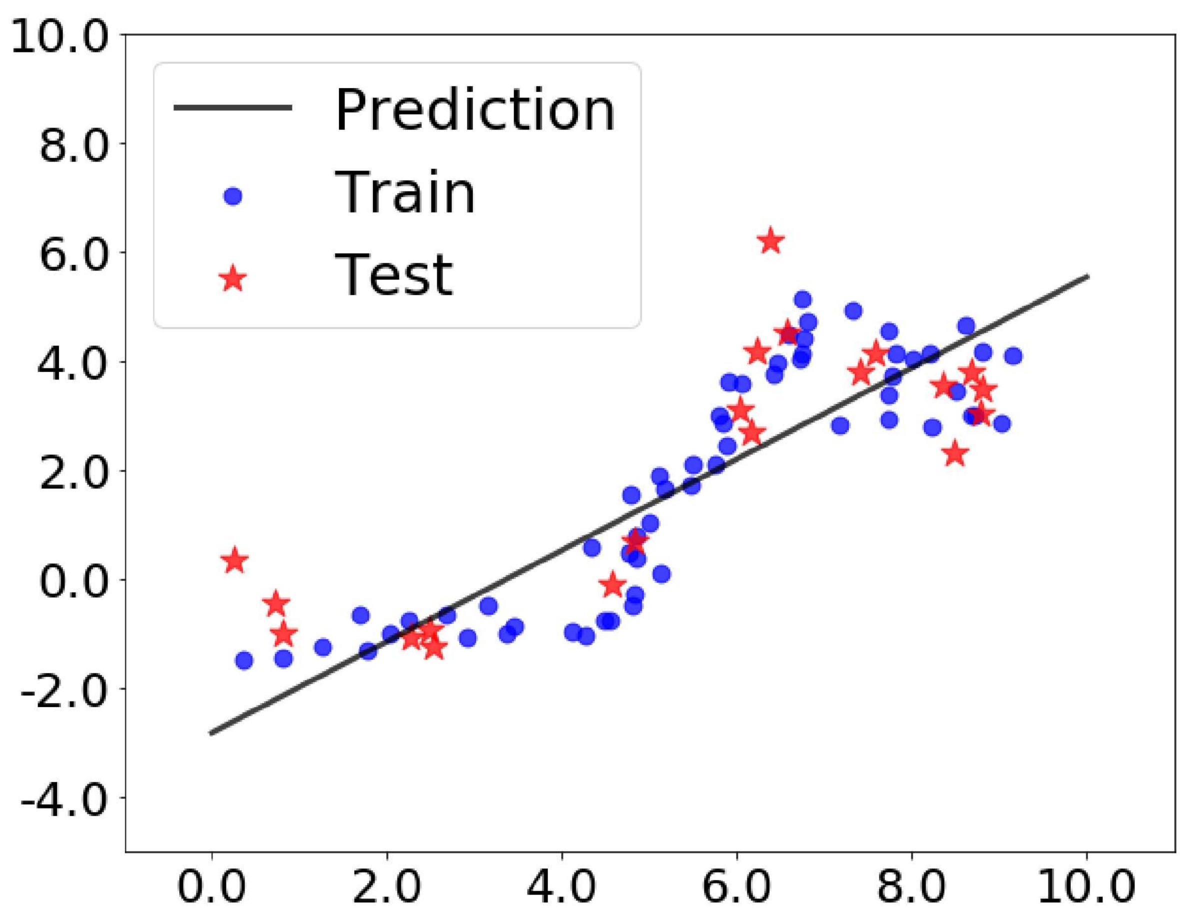
```
Linear Regression Coefficient:  [[-2.82765049  0.83606916]]
linearRegress model, 1 features
[[-2.82765049  0.83606916]]
MSE training data:  1.12771195561
MSE test data:  2.24234920301
```



In [280]:
```
## this next code will create test and train data out of it


#X, Y = ml.shuffleData(X, Y)

#X = X.reshape(-1, 1) # this is to rehape


#Xtr, Xte, Ytr, Yte = ml.splitData(X, Y, 0.75)


# Plotting the data
f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.scatter(Xtr, Ytr, s=80, color='blue', alpha=0.75, label='Train')
ax.scatter(Xte, Yte, s=240, marker='*', color='red', alpha=0.75, label='Test')


ax.set_xlim(-1, 11)
ax.set_ylim(-5, 8)
ax.set_xticklabels(ax.get_xticks(), fontsize=25)
ax.set_yticklabels(ax.get_yticks(), fontsize=25)
plt.show()
```



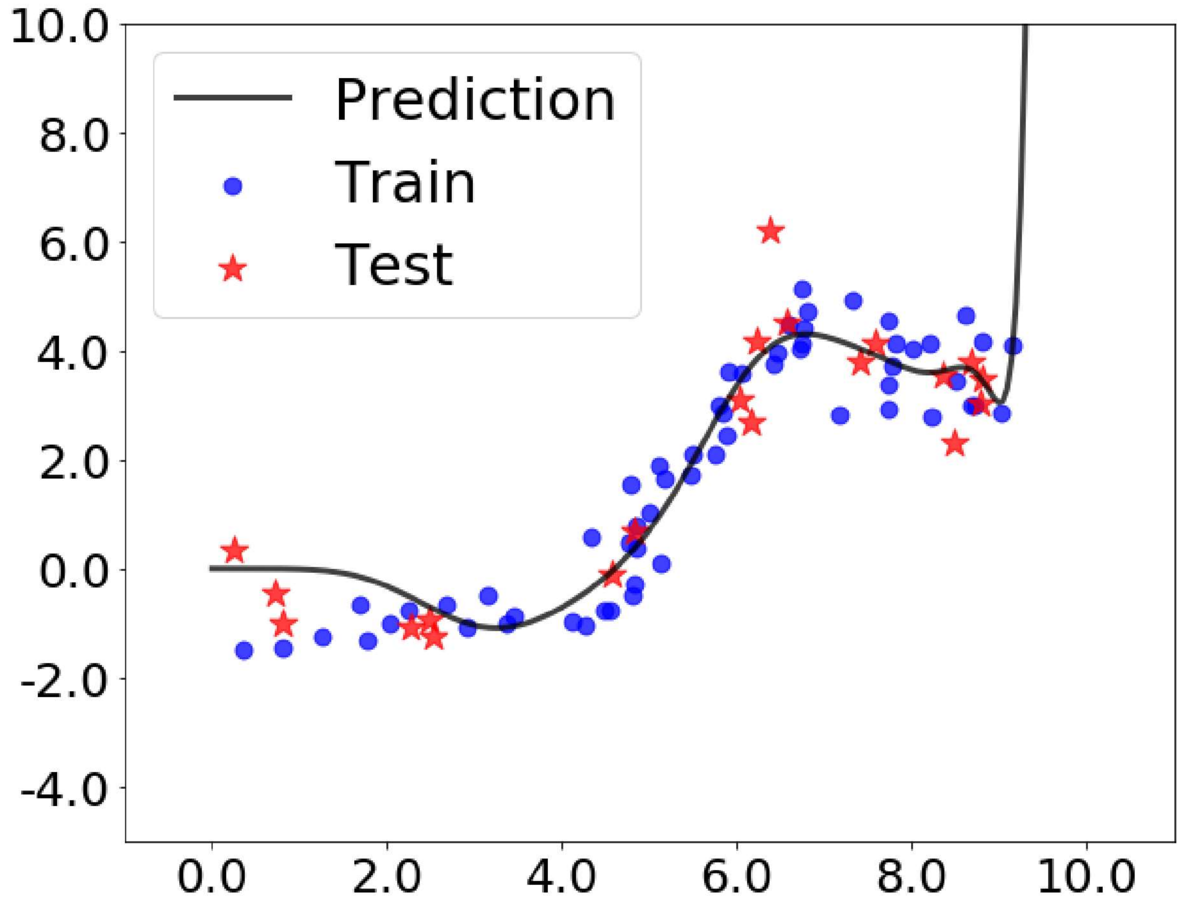
In [281]:
```
# this would add the prediction line.


# Plotting the data
f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.scatter(Xtr, Ytr, s=80, color='blue', alpha=0.75, label='Train')
ax.scatter(Xte, Yte, s=240, marker='*', color='red', alpha=0.75, label='Test')

# Also plotting the regression line
ax.plot(xs, ys, lw=3, color='black', alpha=0.75, label='Prediction')


ax.set_xlim(-1, 11)
ax.set_ylim(-5, 10)
ax.set_xticklabels(ax.get_xticks(), fontsize=25)
ax.set_yticklabels(ax.get_yticks(), fontsize=25)

# Controlling the size of the legend and the location.
ax.legend(fontsize=30, loc=2)

plt.show()
```



In [282]:
```
Xtr2 = np.zeros( (Xtr.shape[0],2) ) # create Mx2 array to store features
Xtr2[:,0] = Xtr[:,0] # place original "x" feature as X1
Xtr2[:,1] = Xtr[:,0]**2 # place "x^2" feature as X2
# Now, Xtr2 has two features about each data point: "x" and "x^2"


#print(Xtr2.shape[0])
# here I decided to slice the list by a third so it
# only showss 20 elements.
#
print(Xtr2[0: int((Xtr2.shape[0]) / 3), :])
```

```
[[  3.4447005   11.86596153]
 [  4.7580645   22.63917779]
 [  6.4170507   41.17853969]
 [  5.7949309   33.58122414]
 [  7.7304147   59.75931143]
 [  7.8225806   61.19276724]
 [  7.7304147   59.75931143]
 [  7.7764977   60.47391648]
 [  8.6751152   75.25762373]
 [  6.4631336   41.77209593]
 [  5.1267281   26.28334101]
 [  6.7396313   45.42263006]
 [  3.1451613    9.8920396 ]
 [  9.1589862   83.88702821]
 [  8.2373272   67.8535594 ]
 [  4.8041475   23.0798332 ]
 [  0.35714286   0.12755102]
 [  8.0069124   64.11064618]
 [  2.2465438    5.04695905]
 [  6.7626728   45.7337434 ]]
```

In [283]:
```
degree = 18
XtrP = ml.transforms.fpoly(Xtr, degree, False)

lr = ml.linear.linearRegress(XtrP, Ytr)

# Make sure you use the currect space.
xs = np.linspace(0, 10, 200)
xs = np.atleast_2d(xs).T

# Notice that we have to transform the predicting xs too.
xsP = ml.transforms.fpoly(xs, degree, False)
ys = lr.predict(xsP)



# Plotting the data
f, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.scatter(Xtr, Ytr, s=80, color='blue', alpha=0.75, label='Train')
ax.scatter(Xte, Yte, s=240, marker='*', color='red', alpha=0.75, label='Test')

# Also plotting the regression line. in the plotting we plot the xs and not th
e xsP
ax.plot(xs, ys, lw=3, color='black', alpha=0.75, label='Prediction')

ax.set_xlim(-1, 11)
ax.set_ylim(-5, 10)
ax.set_xticklabels(ax.get_xticks(), fontsize=25)
ax.set_yticklabels(ax.get_yticks(), fontsize=25)

# Controlling the size of the legend and the location.
ax.legend(fontsize=30, loc=0)

plt.show()
```



below I plotted Train models of degree

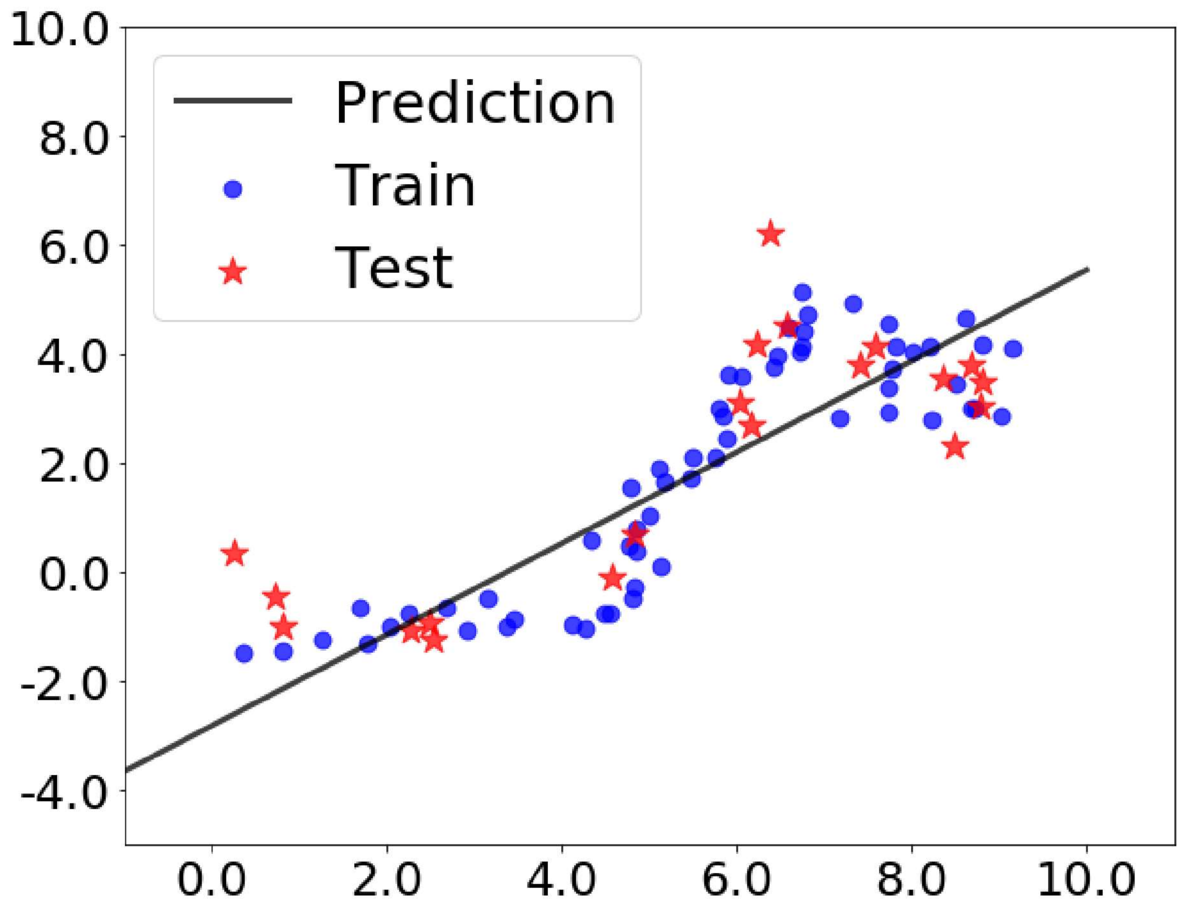
d = 1,3,5,7,10,18

In [284]:

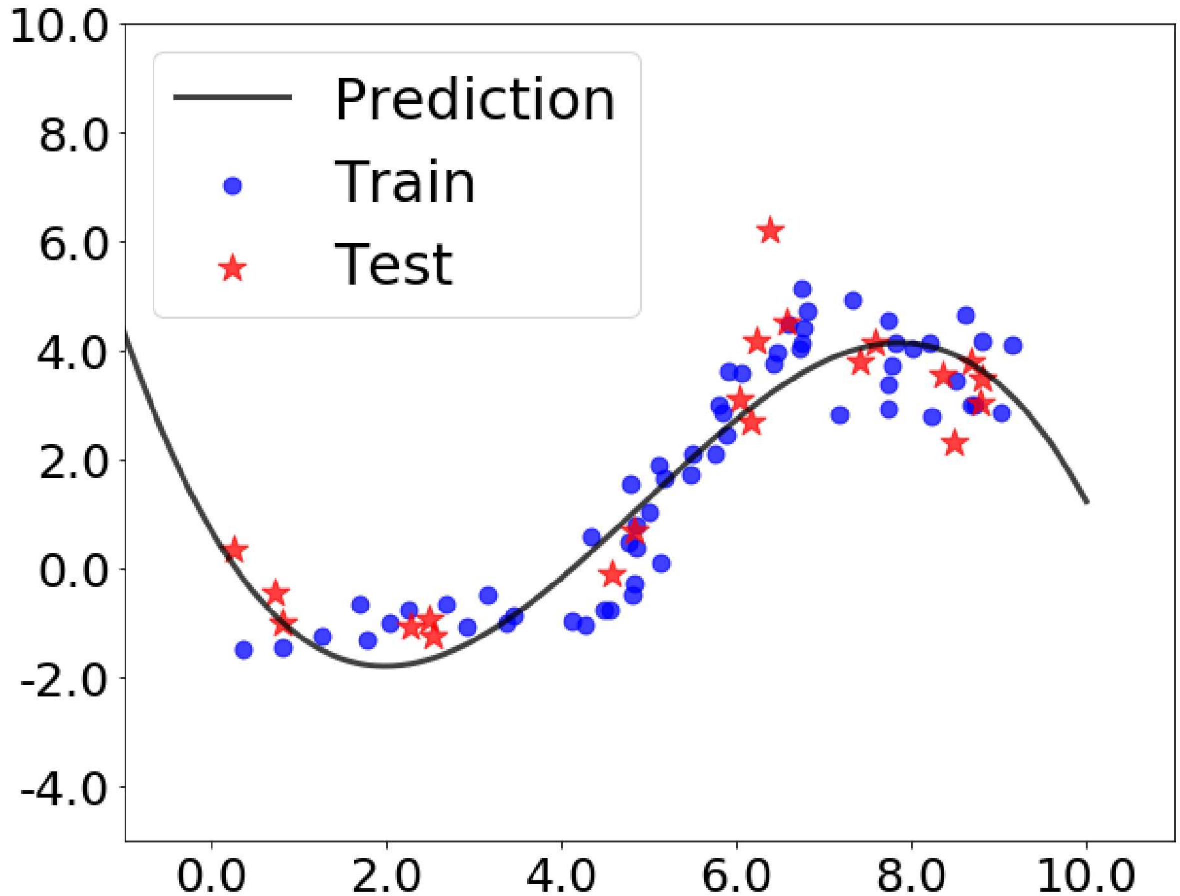
```
#degree = 7


# Plotting the data


degrees = [1,3,5,7,10,18]

for degree in degrees:
    print('Degree: ' , degree)

    # create polynomial up to the degree
    XtrP = ml.transforms.fpoly(Xtr, degree, False)
    # rescale the data matrix so that the features have similar ranges/varienc
e
   # XtrP, params = ml.transforms.rescale(XtrP)
    # params returns the transformation parameter (shift & scale)
    # then we train the model on the scaled feature matrix
    lr = ml.linear.linearRegress(XtrP, Ytr) #create and train model
    # apply the same poly expansion & scaling to Xtest
    #XteP,_ = ml.transforms.rescale(ml.transforms.fpoly(Xte, degree, False), p
```

```
arams)


    # Make sure you use the currect space.
    xs = np.linspace(-10, 10, 200)
    xs = np.atleast_2d(xs).T
    # Notice that we have to transform the predicting xs too.
    xsP = ml.transforms.fpoly(xs,degree, False)
    ys = lr.predict(xsP)

    f, ax = plt.subplots(1, 1, figsize=(10, 8))

    ax.scatter(Xtr, Ytr, s=80, color='blue', alpha=0.75, label='Train')
    ax.scatter(Xte, Yte, s=240, marker='*', color='red', alpha=0.75, label='Te
st')
    #ax.scatter(Xte, YteHat, s=80, marker='D', color='forestgreen', alpha=0.7
5, label='Yhat')


    # Also plotting the regression line. in the plotting we plot the xs and no
t the xsP
    ax.plot(xs, ys, lw=3, color='black', alpha=0.75, label='Prediction')

    ax.set_xlim(-1, 11)
    ax.set_ylim(-5, 10)
    ax.set_xticklabels(ax.get_xticks(), fontsize=25)
    ax.set_yticklabels(ax.get_yticks(), fontsize=25)

    # Controlling the size of the legend and the location.
    ax.legend(fontsize=30, loc=0)





    plt.show()
```

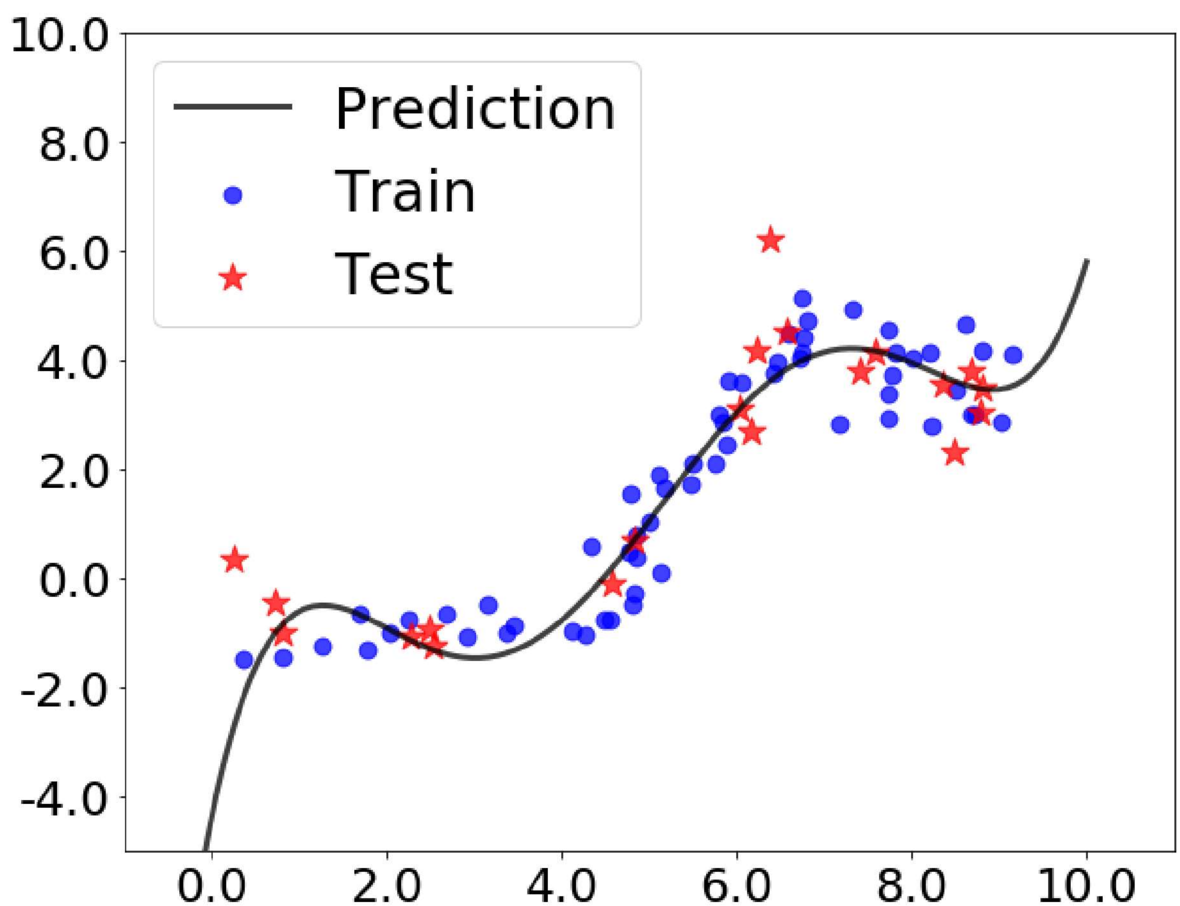
Degree: 1



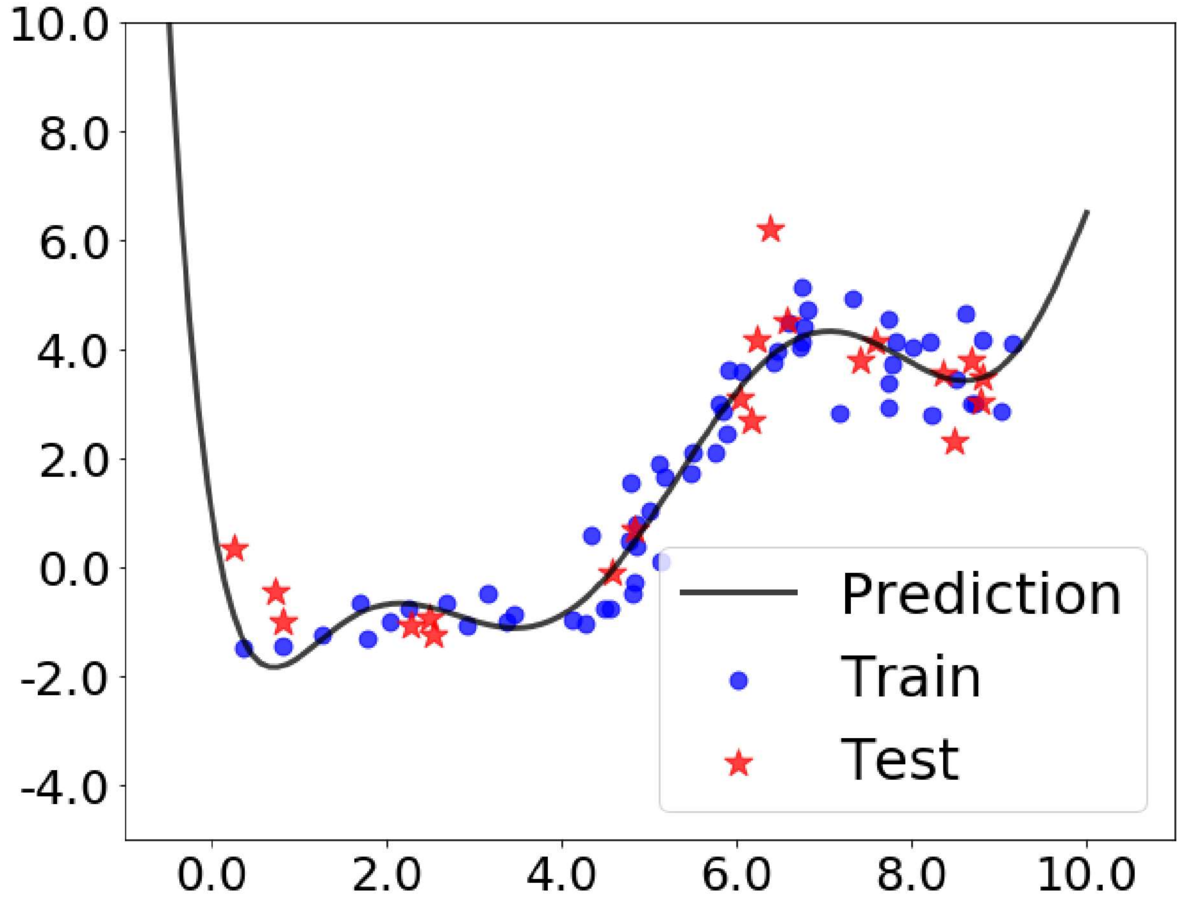
Degree: 3


Prediction
Train
Test
10.0
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
0.0
2.0
4.0
6.0
8.0
10.0

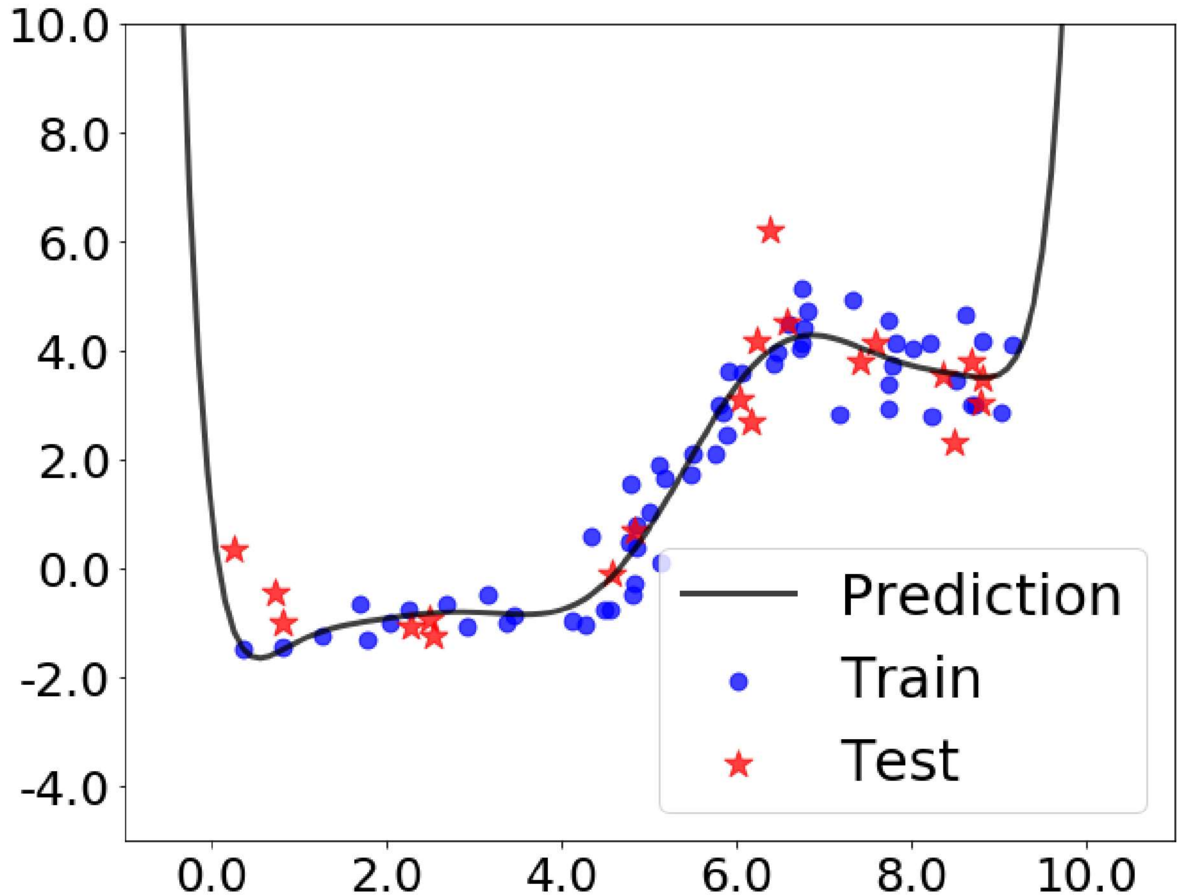
```
Degree:  5
```


Prediction
Train
Test
10.0
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
0.0
2.0
4.0
6.0
8.0
10.0

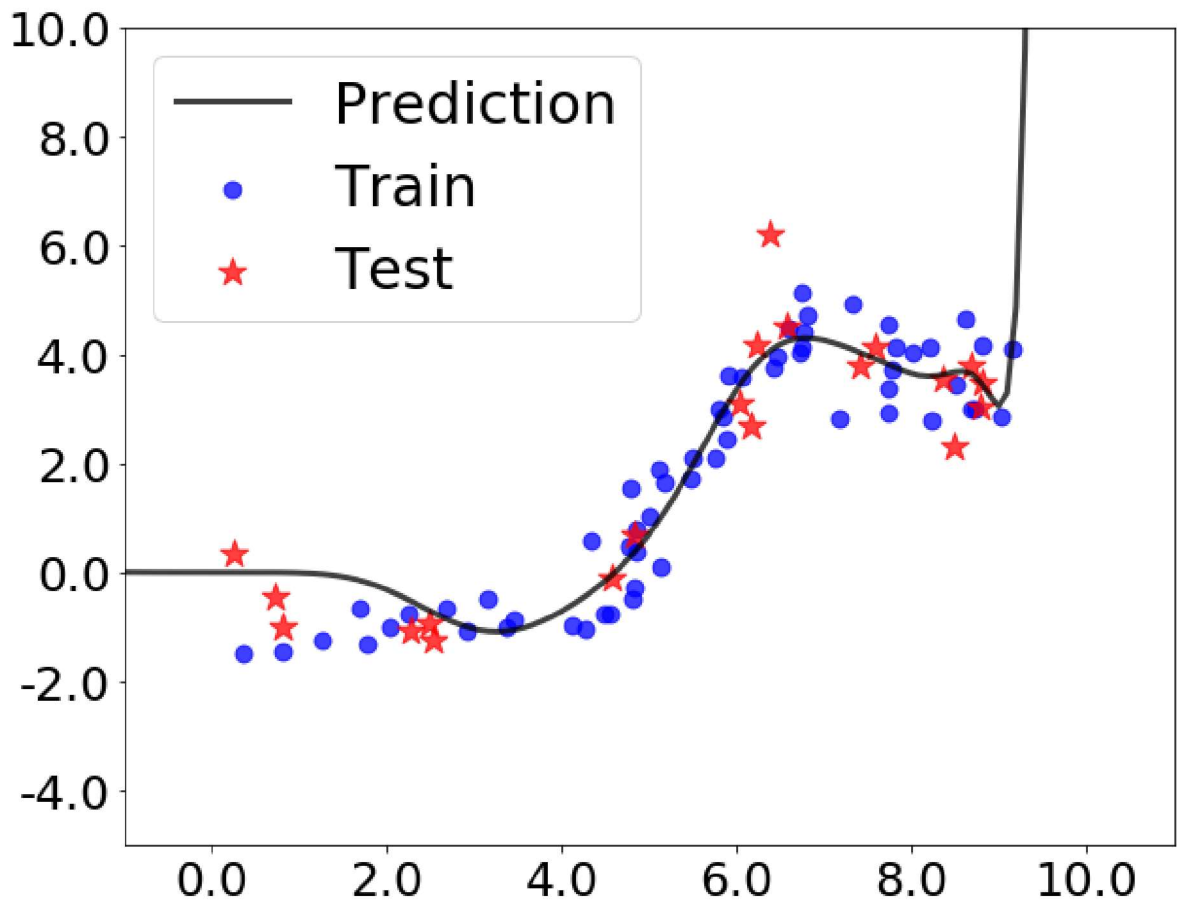
Degree: 7



Degree: 10


10.0
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
0.0
2.0
4.0
6.0
8.0
10.0
Prediction
Train
Test

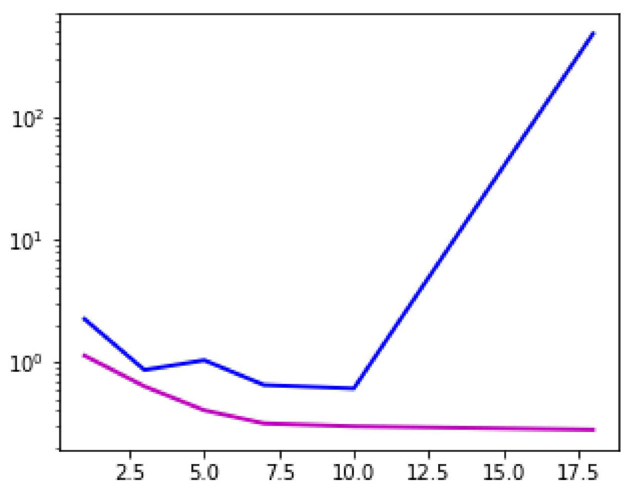
```
Degree:  18
```


10.0
8.0
6.0
4.0
2.0
0.0
-2.0
-4.0
0.0
2.0
4.0
6.0
8.0
10.0
Prediction
Train
Test

Plotting the training and test errors on a log scale as a fnx of a degree

In [287]:
```
import mltools.transforms as xform
Phi = lambda X: xform.rescale(xform.fpoly(X,degrees[d],bias=False),params)[0]

def MSE(Y, Yhat):
    return np.mean((Y-Yhat)**2)
degrees = np.array([1,3,5,7,10,18])
lr = [ [] ]*6

errs = []
errTrain = np.zeros((6,))
errTest = np.zeros((6,))


for d in range(len(degrees)):
    XtP = xform.fpoly(Xtr, degrees[d], bias=False) #
    XtP,params = xform.rescale(XtP) # normalize scale & save transform paramet
ers
    lr[d] = ml.linear.linearRegress( Phi(Xtr), Ytr , reg=1e-1000)
    errTrain[d] = lr[d].mse(Phi(Xtr),Ytr)
    errTest[d] = lr[d].mse(Phi(Xte),Yte)

plt.show()

#print(learners)

print(errTrain)
print(errTest)

#print(degrees)
plt.rcParams['figure.figsize'] = (5.0, 4.0)
plt.semilogy(degrees,errTrain,'m-',degrees,errTest,'b-',linewidth=2);
```

```
[ 1.12771196  0.63396521  0.40424895  0.31563467  0.29894798  0.28050321]
[   2.2423492     0.86161148    1.03441902    0.65022461    0.60906007
  481.20279118]
```

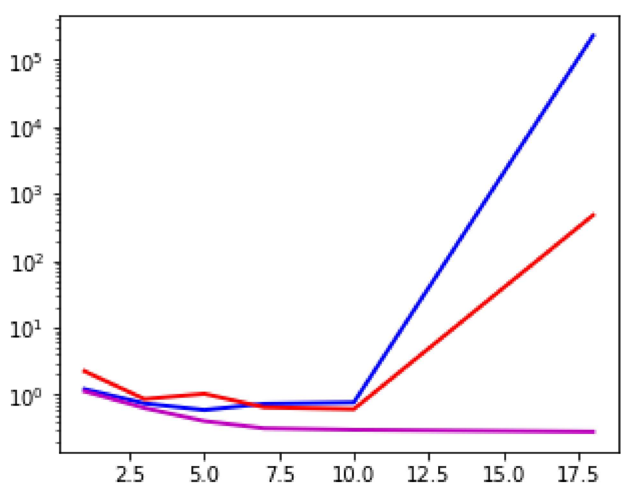


Lookig at the plot above we see that degree 7 or 10 would be the lowest validation error. I would recommend any polynomial between 7-10.

Problem 2 - Cross-Validation

In [289]:
```
nFolds = 5
Phi = lambda X: xform.rescale(xform.fpoly(X,degrees[d],bias=False),params)[0]

degrees = np.array([1,3,5,7,10,18])

errVal = np.zeros((degrees.shape[0], 5)) # a 6*5 array




# Run over all degrees, and each fold for each degree
for d in range(len(degrees)):
    for iFold in range(nFolds):
        [Xti,Xvi,Yti,Yvi] = ml.crossValidate(Xtr,Ytr,nFolds,iFold)

        XtiP = xform.fpoly(Xti, degrees[d], bias=False)
        XtiP,params = xform.rescale(XtiP)
        lr = ml.linear.linearRegress(Phi(Xti),Yti, reg= 1e-1000)

        errVal[d,iFold] = lr.mse(Phi(Xvi),Yvi )



errVal = np.mean(errVal, axis=1)


print(errVal)

plt.rcParams['figure.figsize'] = (5.0, 4.0)
plt.semilogy(degrees,errVal,'b-', linewidth = 2)

plt.semilogy(degrees,errTrain,'m-',degrees,errTest,'r-',linewidth=2);




plt.show()
```

```
[  1.21186266e+00   7.42900575e-01   5.91070373e-01   7.33563783e-01
   7.67705686e-01   2.31535671e+05]
```



In [292]:
```
print(errVal)
print(errTest)
print(errTrain)
```

```
[  1.21186266e+00   7.42900575e-01   5.91070373e-01   7.33563783e-01
   7.67705686e-01   2.31535671e+05]
[   2.2423492     0.86161148    1.03441902    0.65022461    0.60906007
  481.20279118]
[ 1.12771196  0.63396521  0.40424895  0.31563467  0.29894798  0.28050321]
```

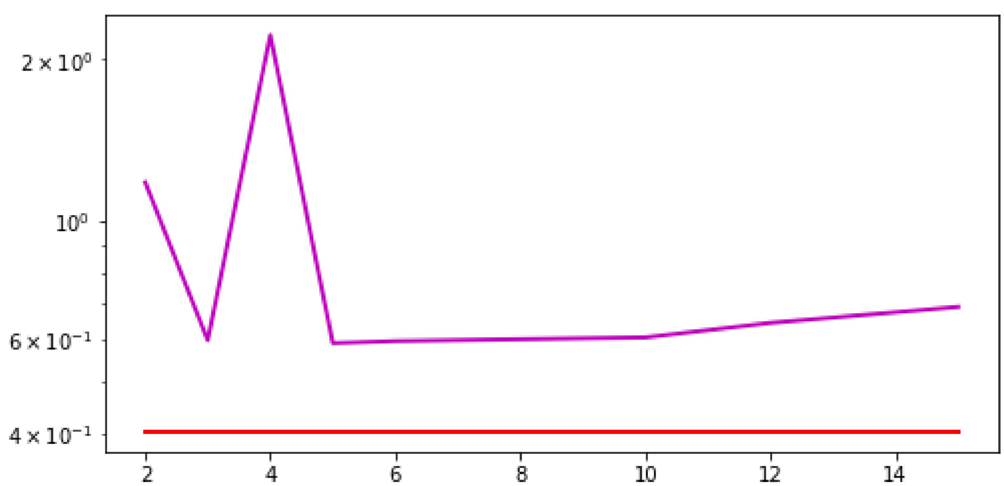
In the code snippet above I printed the MSE for the actual test to compare with the MSE with cross validation. I notice that degree 5, 7 and 10 are the ones closed to each other on the mean squre error. Overall they are both close to each other, however the actual test on degree 18 had a 481 MSE

In [290]:
```
print(min(errVal))
```

```
0.591070372641
```

I would recommend degree 5 based on five fold cross validation. It is the has the minimal MSE which is .5910

In [297]:
```
def get_mean(x):
    return np.mean(x)


degree, Folds_lst = (5,  [2,3,4,5,6,10,12,15])
mse_errorX, mse_errorV = ([], [])

for nFold in Folds_lst:
    mse_errorX2, mse_errorV2 = ([], [])

    for iFold in range(nFold):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,nFold,iFold)

        XtiP = xform.fpoly(Xti, degree, bias=False)
        XtiP,params = xform.rescale(XtiP)
        Phi = lambda X: xform.rescale(xform.fpoly(X,degree,bias=False),params)
[0]

        lv = ml.linear.linearRegress(Phi(Xtr),Ytr)
        mseV = lv.mse(Phi(Xtr),Ytr )
        mse_errorV2.append(mseV)

        lx = ml.linear.linearRegress(Phi(Xti),Yti)
        mseX = lx.mse(Phi(Xvi),Yvi)
        mse_errorX2.append(mseX)

    mse_errorV.append(get_mean(mse_errorV2))
    mse_errorX.append(get_mean(mse_errorX2))


f, ax = plt.subplots(1,1, figsize = (8, 4))
ax.semilogy(Folds, mse_errorX,'m-', Folds, mse_errorV, 'r-', linewidth = 2)
plt.show()
```



here we see that the training vs validation when I have degree 5 which is the ideal polynomial. The training as the folds increase it remains with low MSE, however the validation spikes up between 3 and 6 and then remains constant after 6 folds.

Statement of Collaboration

Troughout the code snippet I provided as much infotmation needed. With the code snippet provided by the professor/TA I was able to implement the information needed to accomplish the homework problems. Also, with step by step in how I am obtaining my answers. Going over lecture notes and with the help of google search I was able to understand cross validation and was able to compute the information needed. I was trying to be as detailed as possible with comments and labeling plots. I also utilized piazza in to have an idea in how to plot and calculate the five fold cross-validation, which I found the most challenging part of this assignment.