273A hw2

EnYu Huang

January 27, 2021

1 CS 273A Homework 2

1.1 Homework1 4.4

In joint Bayes classifier we have 2^5 parameters for each y (y = 1 or -1) to compute, however in naive Bayes classifier, we only have to look up the table in problem 4.1, which is much less than joint Bayes classifier, also the more complex the model is, it might cause overfitting in the end, therefore there is no need to use joint model for these data.

1.2 Homework1 4.5

The naive Bayes model over features from x2 to x5: P(X = x2, x3, x4, x5) | y=1)P(y=1) / (P(X = x2, x3, x4, x5) | y=1)P(y=1) + P(X = x2, x3, x4, x5) | y=-1)P(y=-1)) , from above equation we can know that the lossing of P(x1) will not prevent us from making prediction, hence, there is no need to re-train the model. The parameters in this question are P(xi=0 | y=1),P(xi=1 | y=-1), i= 2 to 5

1.3 Problem 1: Linear Regression

1.3.1 1.

[33]:
```python
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
data = np.genfromtxt("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-dimensional
Y = data[:,1] # doesn't matter for Y
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75) # split data set 75/25
print("Xtr:",Xtr.shape,"Xte:",Xte.shape,"Ytr:",Ytr.shape,"Yte:",Yte.shape)
```

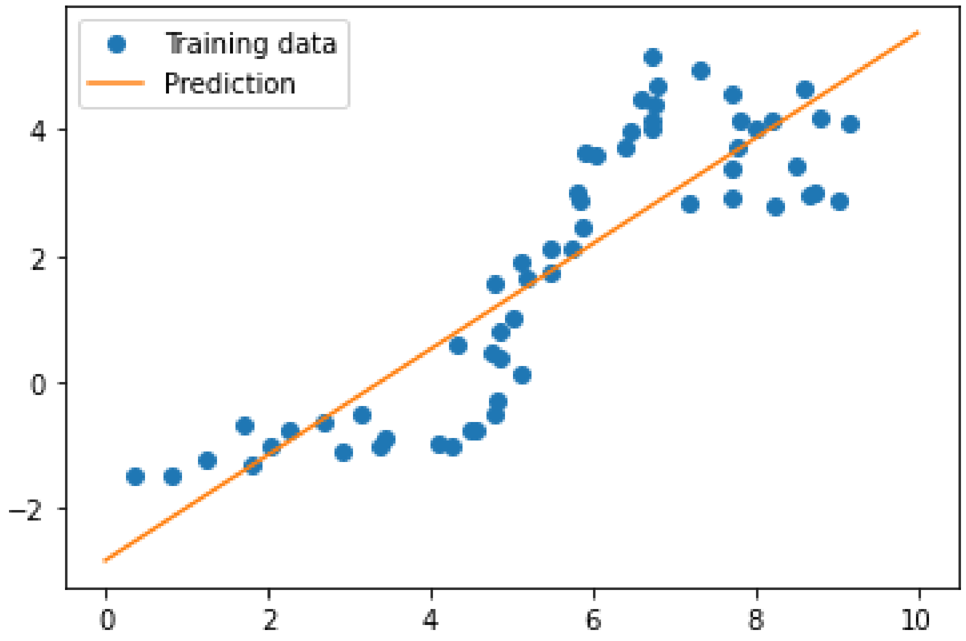
```
Xtr: (60, 1) Xte: (20, 1) Ytr: (60,) Yte: (20,)
```

1.4 2.

1.5 (a)

```
[34]: 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 ourcode)

      ys = lr.predict( xs ) # make predictions at xs

      plt.plot(Xtr,Ytr,"o",label="Training data")
      plt.plot(xs,ys,"-",label="Prediction")
      plt.legend()
```

```
[34]: <matplotlib.legend.Legend at 0x1c52137aac0>
```



1.6 (b)

```
[35]: theta0, theta1 = lr.theta[0][0], lr.theta[0][1]
      print("theta0:",theta0,"theta1:",theta1)
```

```
theta0: -2.827650487664813 theta1: 0.8360691602619539
```

1.7 (c)

[36]:
```
def MSE(X, Y):

    Yhat = lr.predict(X)
    Ys = np.atleast_2d(Y).T
    error = Yhat - Ys
    return(sum(error**2)/len(X))

print ("MSE of training data:",MSE(Xtr, Ytr),"MSE of testing data:",MSE(Xte,␣
 ↪Yte))
```

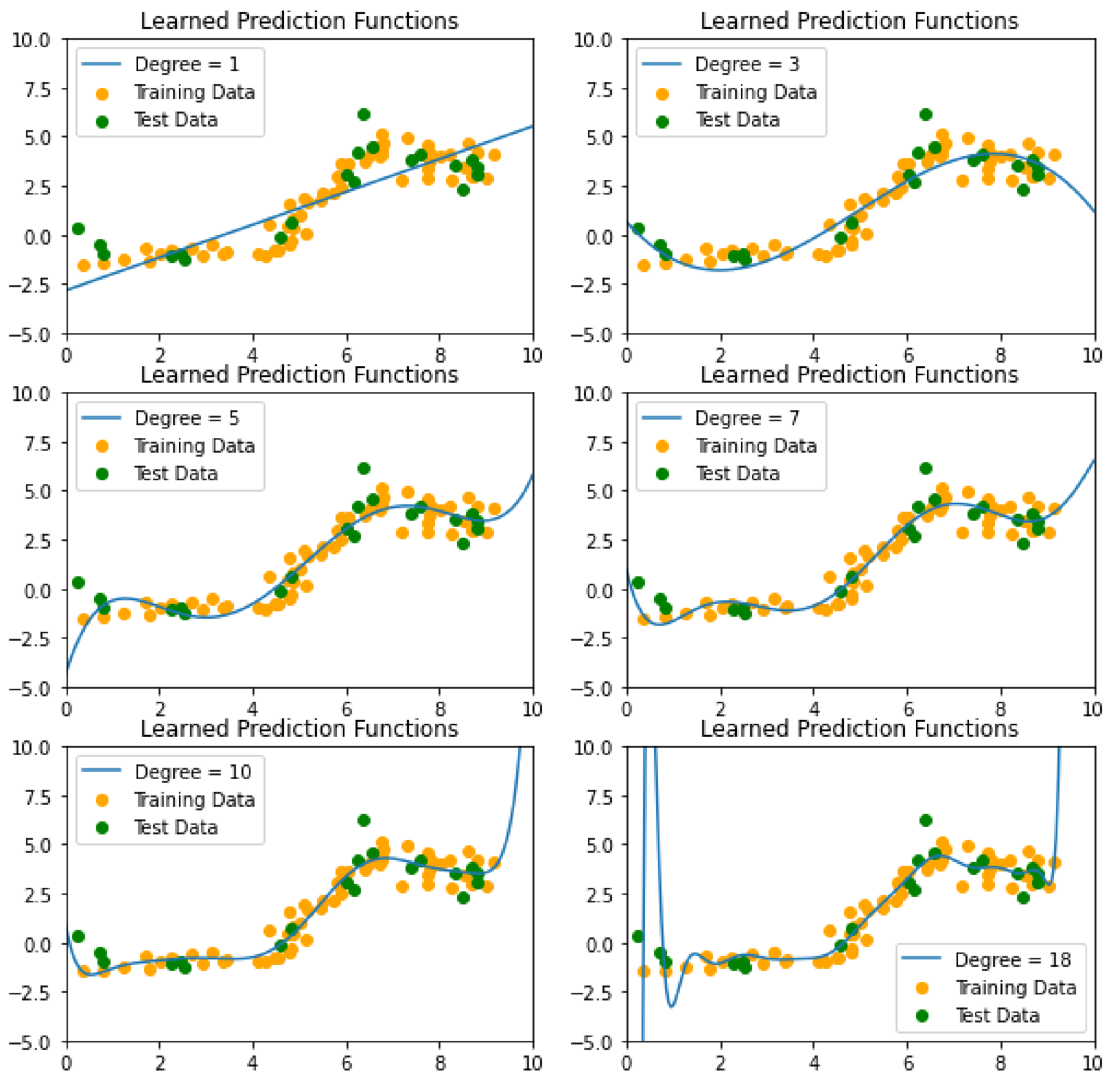
```
MSE of training data: [1.12771196] MSE of testing data: [2.2423492]
```

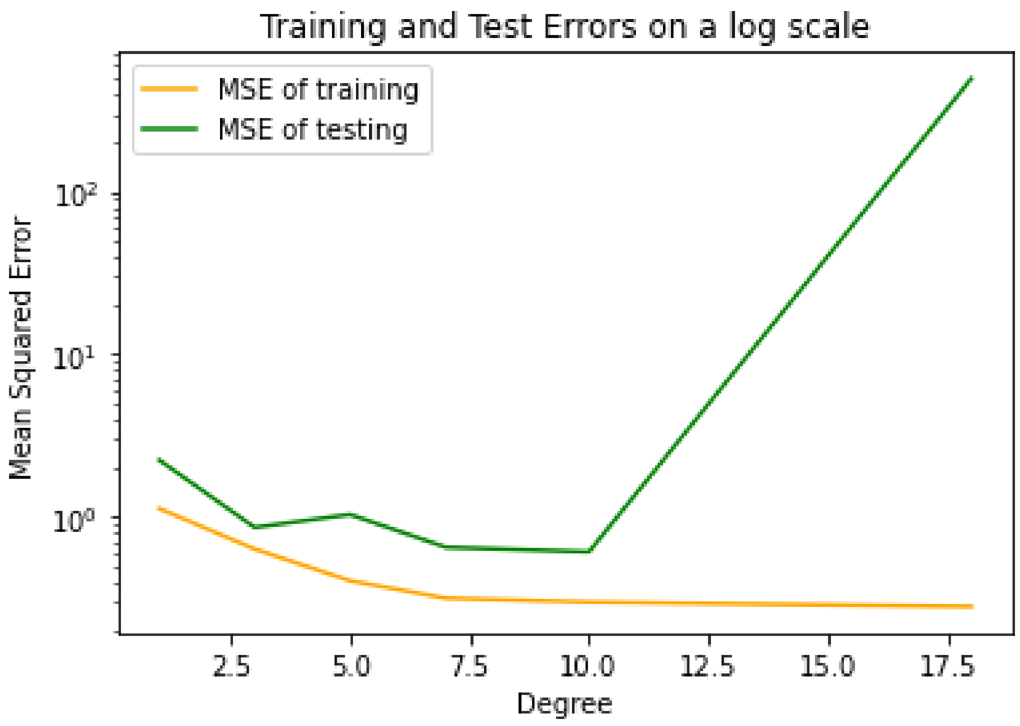
1.8 3. (a)(b)

[37]:
```
fig, ax = plt.subplots(3, 2, figsize=(10, 10))
ax = ax.ravel()
d = [1,3,5,7,10,18]
error_train = np.zeros(len(d));
error_test = np.zeros(len(d));
for i, degree in enumerate(d):
    # Create polynomial features and rescale the data matrix so that the␣
 ↪features have similar ranges / variance
    XtrP,params = ml.transforms.rescale(ml.transforms.fpoly(Xtr, degree,␣
 ↪bias=False))
    # Train model
    lr = ml.linear.linearRegress( XtrP, Ytr )
    XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte,degree,False),␣
 ↪params)
    Xsp,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,degree,False), params)
    Ysp = lr.predict(Xsp)
    # calculate error
    error_train[i] = MSE(XtrP,Ytr)
    error_test[i] = MSE(XteP,Yte)
    print('Degree =', degree, )
    print('Training MSE:', error_train[i])
    print('Test MSE:', error_test[i])
    ax[i].scatter(Xtr, Ytr, color='orange', label='Training Data')
    ax[i].scatter(Xte, Yte, color='green', label='Test Data')
    ax[i].plot(xs,Ysp, label = "Degree = %s" %degree)
    ax[i].set_xlim(0, 10)
    ax[i].set_ylim(-5, 10)
    ax[i].set_title("Learned Prediction Functions")
    ax[i].legend()

plt.show()
```

```python
# plot training and test error on log scale
fig, ax = plt.subplots(1,1)
ax.semilogy(d, error_train, color = "orange", label = "MSE of training")
ax.semilogy(d, error_test, color = "green", label = "MSE of testing" )
ax.set_xlabel("Degree")
ax.set_ylabel("Mean Squared Error")
ax.set_title("Training and Test Errors on a log scale")
ax.legend()
plt.show()
```

```
Degree = 1
Training MSE: 1.127711955609391
Test MSE: 2.242349203010125
Degree = 3
Training MSE: 0.6339652063119632
Test MSE: 0.8616114815450061
Degree = 5
Training MSE: 0.4042489464459156
Test MSE: 1.0344190205633543
Degree = 7
Training MSE: 0.3156346739891712
Test MSE: 0.6502246079664455
Degree = 10
Training MSE: 0.2989479796688226
Test MSE: 0.609060074952002
Degree = 18
Training MSE: 0.280502821234544
Test MSE: 497.8152067603799
```


Learned Prediction Functions
Degree = 1
Training Data
Test Data
Learned Prediction Functions
Degree = 3
Training Data
Test Data
Learned Prediction Functions
Degree = 5
Training Data
Test Data
Learned Prediction Functions
Degree = 7
Training Data
Test Data
Learned Prediction Functions
Degree = 10
Training Data
Test Data
Learned Prediction Functions
Degree = 18
Training Data
Test Data
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
2
4
6
8
10



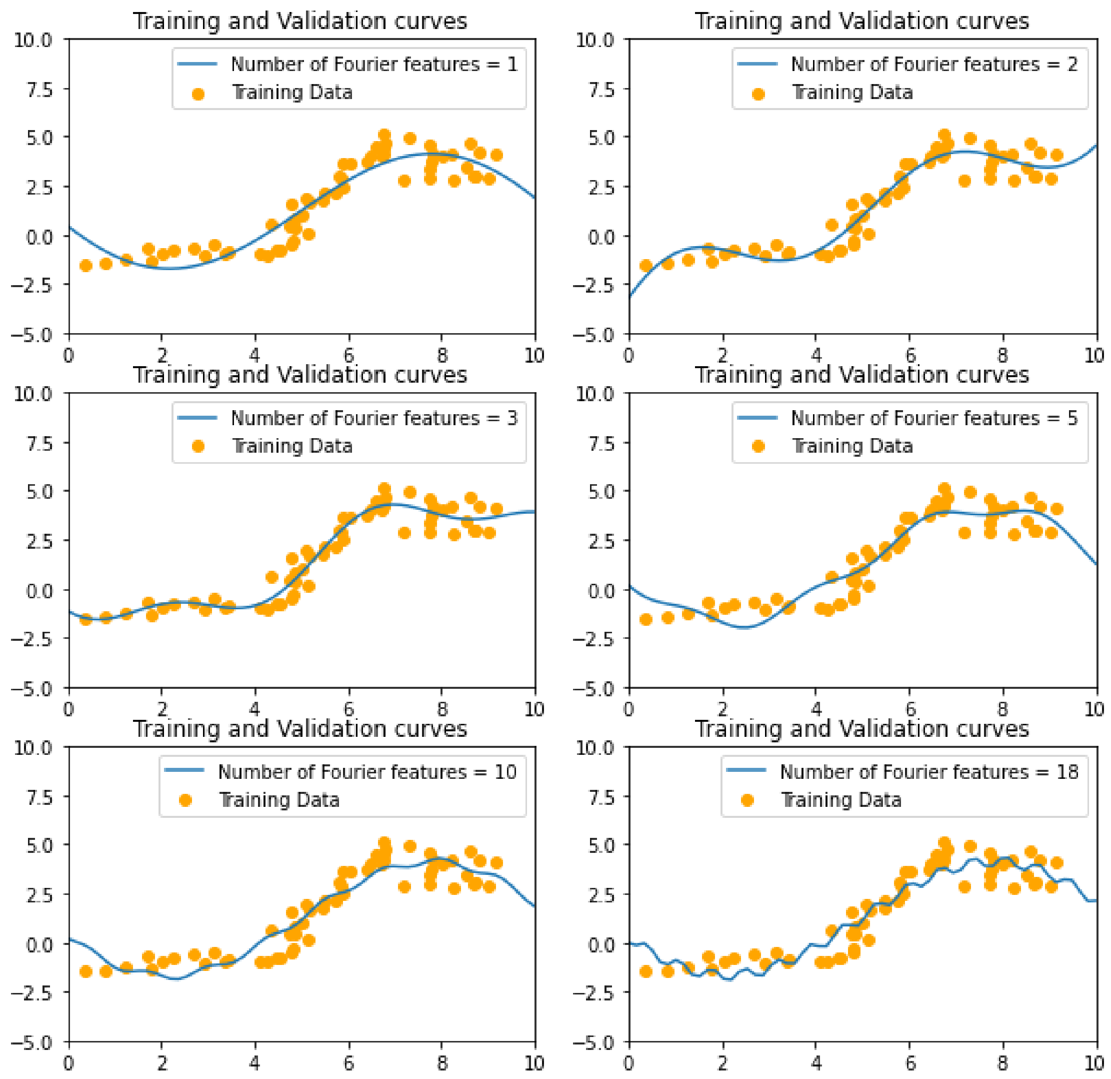
1.9 (c)

Based on the above plot, I recommend the degree of 10 since it has the lowest testing error at this point.

1.10 4.

```
[42]: XtrF = np.zeros( (Xtr.shape[0],5) ) # create Mx5 array to store features
      xsF = np.zeros( (Xtr.shape[0],5) )
      fig, ax = plt.subplots(3,2, figsize=(10, 10))
      xs = np.linspace(0,10,60)
      xs = xs[:,np.newaxis]
      ax = ax.ravel()
      c = [1,2,3,5,10,18]
      for i, cons in enumerate(c):

          XtrF[:,0] = Xtr[:,0] # place original "x" feature as X1
          XtrF[:,1] = np.sin(Xtr[:,0]/2.) # place "sin(x)" feature as X2 (approx.␣
       ↪scaled to# X's range)
          XtrF[:,2] = np.cos(Xtr[:,0]/2.) # place "cos(x)" feature as X3
          XtrF[:,3] = np.sin(Xtr[:,0]*cons/2.) # place "sin(2*x)" feature as X4
          XtrF[:,4] = np.cos(Xtr[:,0]*cons/2.) # place "cos(2*x)" feature as X5
```

```
    xsF[:,0] = xs[:,0]
    xsF[:,1] = np.sin(xs[:,0]/2.)
    xsF[:,2] = np.cos(xs[:,0]/2.)
    xsF[:,3] = np.sin(xs[:,0]*cons/2.)
    xsF[:,4] = np.cos(xs[:,0]*cons/2.)
    lr = ml.linear.linearRegress( XtrF, Ytr )
    ysF = lr.predict(xsF) # make predictions at xs

    ax[i].scatter(Xtr, Ytr, color='orange', label='Training Data')
    ax[i].plot(xs,ysF, label = "Number of Fourier features = %s" %cons)
    ax[i].set_xlim(0, 10)
    ax[i].set_ylim(-5, 10)
    ax[i].set_title("Training and Validation curves")
    ax[i].legend()


plt.show()
```



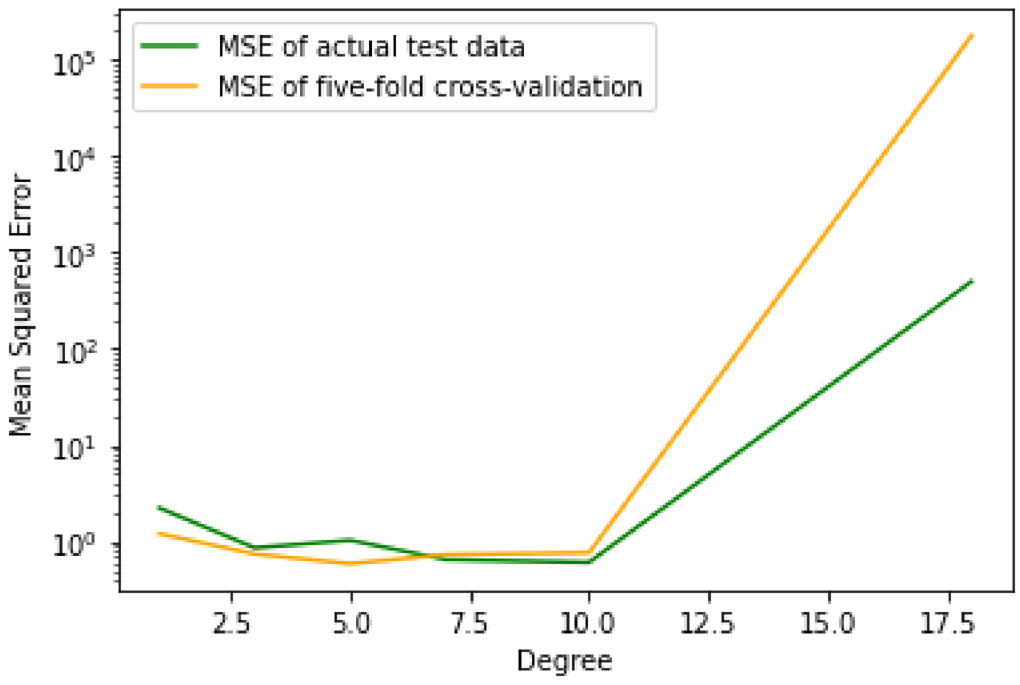
When the number of Fourier Features increase to the value above 5, it will cause over-fitting problem by observing, hence we can conclude that expanding the number of Fourier Features is not always the best solution.

1.11 Problem 2: Cross-validation

1.12 1.

[39]:
```
nFolds = 5;
J = np.zeros(nFolds)
mse = np.zeros(len(d))

for i,degree in enumerate(d):
    for iFold in range(nFolds):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,nFolds,iFold)
        XtiP,params = ml.transforms.rescale(ml.transforms.fpoly(Xti, degree,␣
 ↪bias=False))
        lr = ml.linear.linearRegress(XtiP, Yti)
        XviP,_ = ml.transforms.rescale(ml.transforms.fpoly(Xvi, degree,␣
 ↪bias=False), params)
        J[iFold] = MSE(XviP,Yvi)
    mse[i] = np.mean(J)
    print('degree =', degree)
    print("error:", mse[i])

fig, ax = plt.subplots()
ax.set_xlabel("Degree")
ax.set_ylabel("Mean Squared Error")
ax.semilogy(d, error_test, color = "green", label = "MSE of actual test data" )
ax.semilogy(d, mse,color = "orange",label = "MSE of five-fold cross-validation")
ax.legend()
plt.show()
```

```
degree = 1
error: 1.2118626629641984
degree = 3
error: 0.7429005752051646
degree = 5
error: 0.5910703726407939
degree = 7
error: 0.7335637831327384
degree = 10
error: 0.7677056830803561
degree = 18
error: 176227.52604921878
```



1.13 2.

The Mean Squared Error of five-fold cross-validation is lower than actual test data when degree < 5 (approximately), after degree = 10, the MSE curve of five-fold cross-validation goes up steeply much more than actual test data

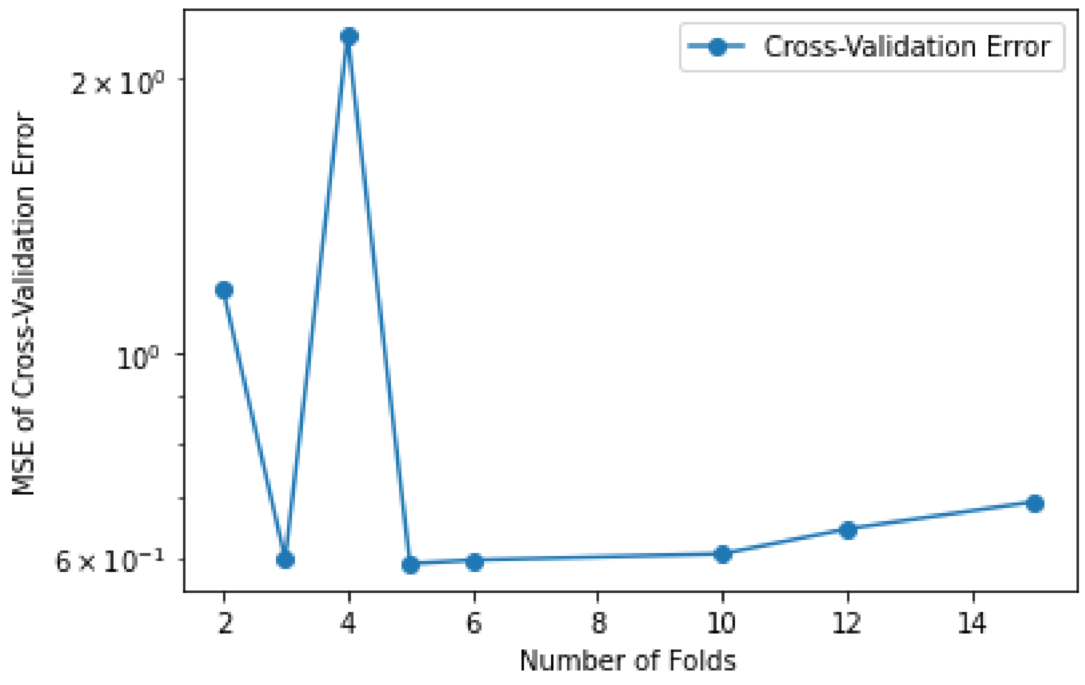
1.14 3.

The degree of 5 will be the ideal choice in this case, from the plot in problem 1, we can observe that the testing error is lowest at this point.

1.15 4.

[40]:
```
nFolds = [2,3,4,5,6,10,12,15]
degree = 5 # For what we choose previously
mse = np.zeros(len(nFolds))

for j, Fold in enumerate(nFolds):
    J = np.zeros(Fold)

    for i in range(Fold):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,Fold,i)
        XtiP,params = ml.transforms.rescale(ml.transforms.fpoly(Xti, degree,␣
↪bias=False))
        lr = ml.linear.linearRegress(XtiP, Yti)
        XviP,_ = ml.transforms.rescale(ml.transforms.fpoly(Xvi, degree,␣
 ↪bias=False), params)
        J[i] = MSE(XviP, Yvi)

    mse[j] = np.mean(J)
    print('Fold =', Fold)
    print("cross-validation error:", mse[j])
fig, ax = plt.subplots()
ax.semilogy(nFolds, mse, marker ='o',label = "Cross-Validation Error")
ax.set_xlabel("Number of Folds")
ax.set_ylabel("MSE of Cross-Validation Error")
plt.legend()
plt.show()
```

```
Fold = 2
cross-validation error: 1.1795458641319012
Fold = 3
cross-validation error: 0.5984555010979001
Fold = 4
cross-validation error: 2.219526156064467
Fold = 5
cross-validation error: 0.5910703726407939
Fold = 6
cross-validation error: 0.5963380050012015
Fold = 10
cross-validation error: 0.6058256908836851
Fold = 12
cross-validation error: 0.6448758386950425
Fold = 15
cross-validation error: 0.690566966174373
```



At nFold = 2 the MSE of Cross-Validation starts at high point, later on, it has a slightly decreasing then curve surging at nFold = 4 again to the new peak of MSE that close to 2, the unusual phenomenon at nFold = 4 can be explained as the first three blocks of training data exist huge difference from the last block with testing data, hence the error will surge or decline abnormally. When the nFold > 4, the curve decline to 0.6 then inrceasing gradually along with the numbers of Fold.

1.16 Statement of Collaboration

I finished this assignment alone and only discussed the concept of extra credit with Josh Ho. Below is the resource I used to have a better understanding with problem 2 https://www.youtube.com/watch?v=fSytzGwwBVw .