# Homework2_178

October 20, 2020

1a.

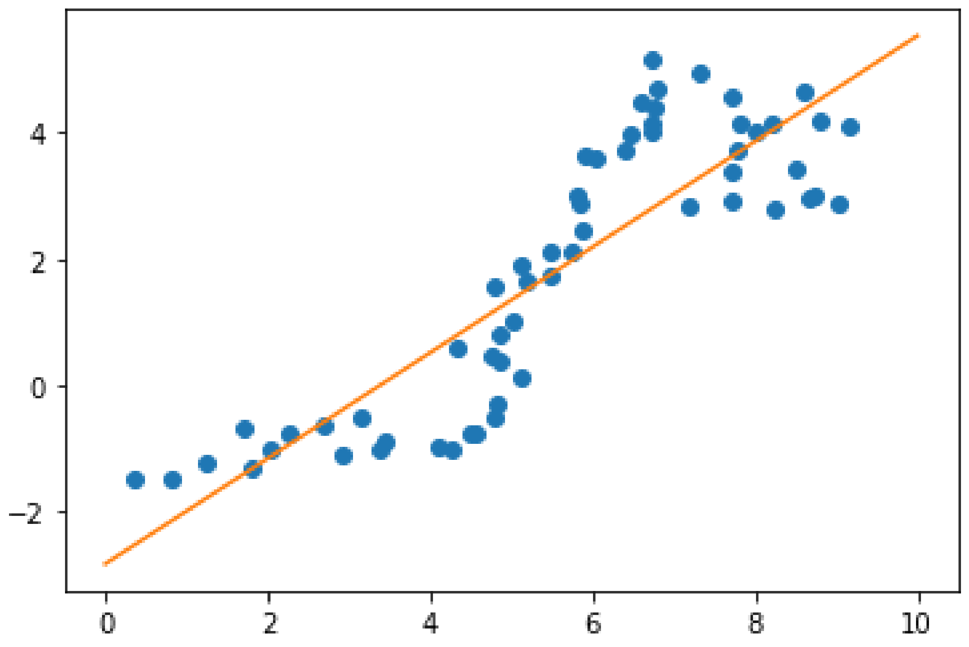
```
[122]: import numpy as np
       import matplotlib.pyplot as plt
       import mltools as ml

       data = np.genfromtxt("data/curve80.txt",delimiter=None) # load the text file
       X = data[:,0]
       X = np.atleast_2d(X).T # code expects shape (M,N) so make sure it's␣
        ↪2-dimensional
       Y = data[:,1]
       Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75) # split data set 75/25

       print(np.shape(Xtr))
       print(np.shape(Xte))
       print(np.shape(Ytr))
       print(np.shape(Yte))
```

```
(60, 1)
(20, 1)
(60,)
(20,)
```

2a.

```
[123]: 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

       plt.plot(Xtr,Ytr, 'o')
       plt.plot(xs, ys)
```

```
[123]: [<matplotlib.lines.Line2D at 0x11b6bacd0>]
```



2b.

```
[124]: print(lr.theta)
```

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

2c.

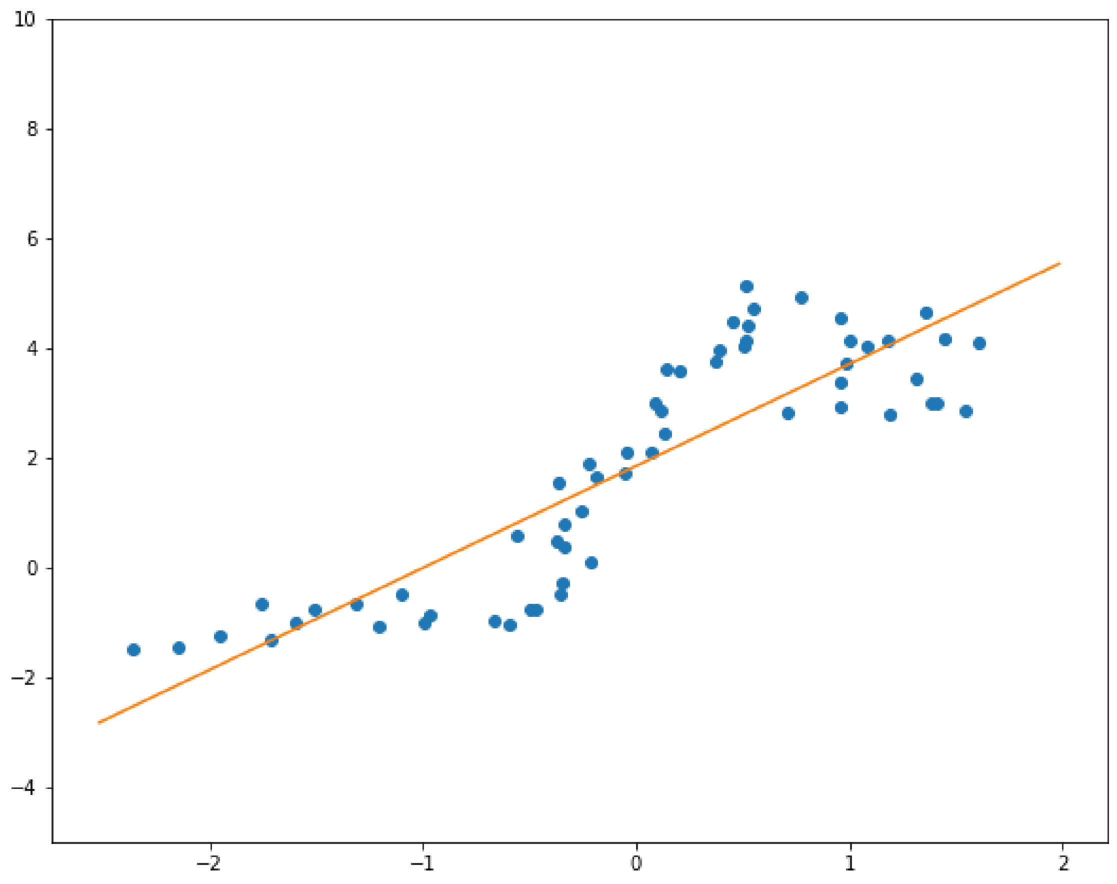
```
[125]: summation = 0
       for x in range(len(Xtr)):
           Ypredict = lr.theta[0][1] * Xtr[x] + lr.theta[0][0]
           summation += pow((Ypredict - Ytr[x]), 2)
       print("Mean squared error on training data: ", summation/len(Xtr))

       summation2 = 0
       for x in range(len(Xte)):
           Ypredict = lr.theta[0][1] * Xte[x] + lr.theta[0][0]
           summation2 += pow((Ypredict - Yte[x]), 2)
       print("Mean squared error on testing data: ", summation/len(Xte))
```

```
Mean squared error on training data:  [1.12771196]
Mean squared error on testing data:  [3.38313587]
```

3a. d = 1

```
[126]: # Create polynomial features up to "degree"; don't create constant feature
       # (the linear regression learner will add the constant feature automatically)
       XtrP = ml.transforms.fpoly(Xtr, 1, 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

#print(Xtr)

# Now, apply the same polynomial expansion & scaling transformation to Xtest:
XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte,1,False), params)

xs = np.linspace(0,10,60)
xs = xs[:,np.newaxis]
xsP,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,1,False), params)

new = lr.predict(xsP)

Yhat = lr.predict(XtrP)
Yhat2 = lr.predict(XteP)

fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
ax.plot(XtrP[:,0], Ytr, 'o') # Plot polynomial regression of desired degree
ax.plot(xsP[:,0], new)
ax.set_ylim(-5, 10) # Set the minimum and maximum limits
plt.show()
```



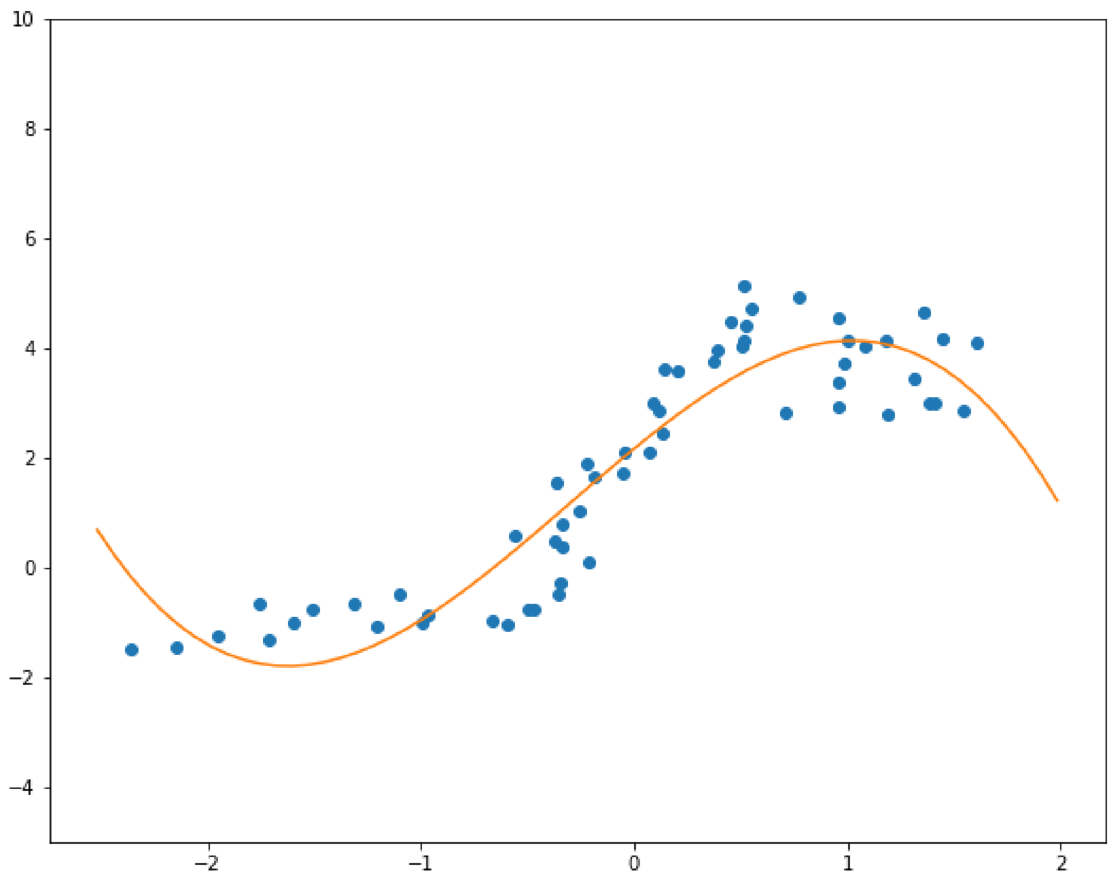
3b. d = 1

```
[127]: sum = 0
for x in range(len(Ytr)):
    sum += pow((Ytr[x] - Yhat[x]), 2)
error = sum/len(Ytr)
print("training error =", error)

sum2 = 0
for x in range(len(Yte)):
    sum2 += pow((Yte[x] - Yhat2[x]), 2)
error2 = sum2/len(Yte)
print("testing error =", error2)
Xtraining = [error]
Xtesting = [error2]
```

```
training error = [1.12771196]
testing error = [2.2423492]
```

3a. d = 3

[128]:
```
# Create polynomial features up to "degree"; don't create constant feature
# (the linear regression learner will add the constant feature automatically)
XtrP = ml.transforms.fpoly(Xtr, 3, 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

#print(Xtr)

# Now, apply the same polynomial expansion & scaling transformation to Xtest:
XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte,3,False), params)

xs = np.linspace(0,10,60)
xs = xs[:,np.newaxis]
xsP,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,3,False), params)

new = lr.predict(xsP)

Yhat = lr.predict(XtrP)
Yhat2 = lr.predict(XteP)

fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
ax.plot(XtrP[:,0], Ytr, 'o') # Plot polynomial regression of desired degree
ax.plot(xsP[:,0], new)
ax.set_ylim(-5, 10) # Set the minimum and maximum limits
plt.show()
```



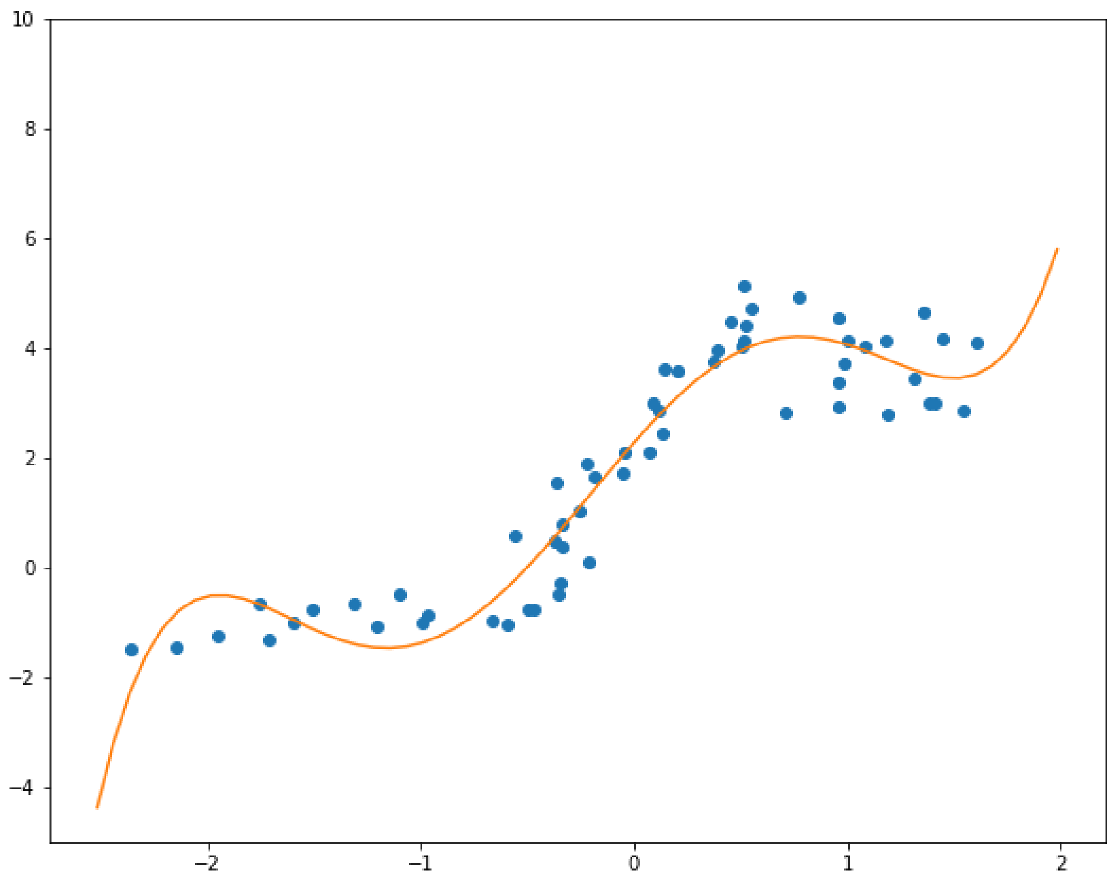
3b. d = 3

```
[129]: sum = 0
       for x in range(len(Ytr)):
           sum += pow((Ytr[x] - Yhat[x]), 2)
       error = sum/len(Ytr)
       print("training error =", error)

       sum2 = 0
       for x in range(len(Yte)):
           sum2 += pow((Yte[x] - Yhat2[x]), 2)
       error2 = sum2/len(Yte)
       print("testing error =", error2)
       Xtraining.append(error)
       Xtesting.append(error2)
```

```
training error = [0.63396521]
testing error = [0.86161148]
```

3a. d = 5

[130]:
```
# Create polynomial features up to "degree"; don't create constant feature
# (the linear regression learner will add the constant feature automatically)
XtrP = ml.transforms.fpoly(Xtr, 5, 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,5,False), params)

xs = np.linspace(0,10,60)
xs = xs[:,np.newaxis]
xsP,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,5,False), params)

new = lr.predict(xsP)

Yhat = lr.predict(XtrP)
Yhat2 = lr.predict(XteP)

fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
ax.plot(XtrP[:,0], Ytr, 'o') # Plot polynomial regression of desired degree
ax.plot(xsP[:,0], new)
ax.set_ylim(-5, 10) # Set the minimum and maximum limits
plt.show()
```



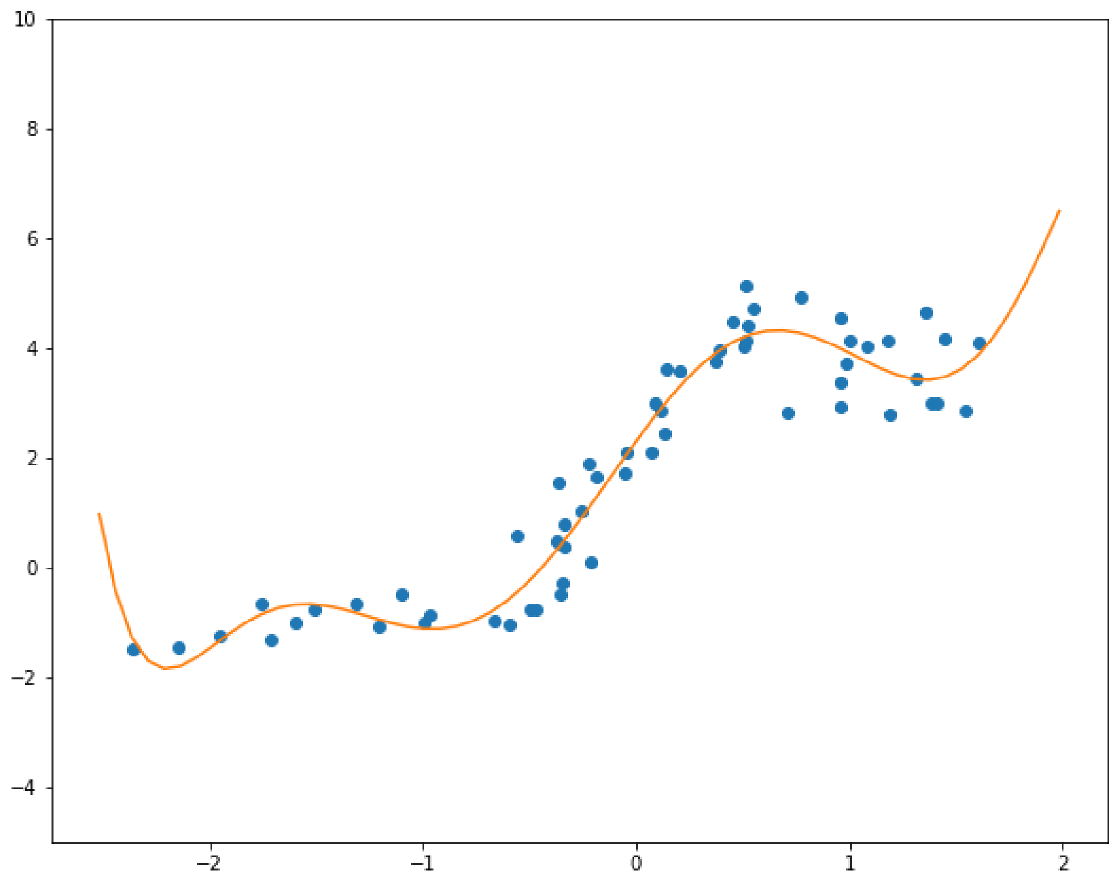
3b. d = 5

```
[131]: sum = 0
       for x in range(len(Ytr)):
           sum += pow((Ytr[x] - Yhat[x]), 2)
       error = sum/len(Ytr)
       print("training error =", error)

       sum2 = 0
       for x in range(len(Yte)):
           sum2 += pow((Yte[x] - Yhat2[x]), 2)
       error2 = sum2/len(Yte)
       print("testing error =", error2)
       Xtraining.append(error)
       Xtesting.append(error2)
```

```
training error = [0.40424895]
testing error = [1.03441902]
```

3a. d = 7

[132]:
```
# Create polynomial features up to "degree"; don't create constant feature
# (the linear regression learner will add the constant feature automatically)
XtrP = ml.transforms.fpoly(Xtr, 7, 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,7,False), params)

xs = np.linspace(0,10,60)
xs = xs[:,np.newaxis]
xsP,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,7,False), params)

new = lr.predict(xsP)

Yhat = lr.predict(XtrP)
Yhat2 = lr.predict(XteP)

fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
ax.plot(XtrP[:,0], Ytr, 'o') # Plot polynomial regression of desired degree
ax.plot(xsP[:,0], new)
ax.set_ylim(-5, 10) # Set the minimum and maximum limits
plt.show()
```



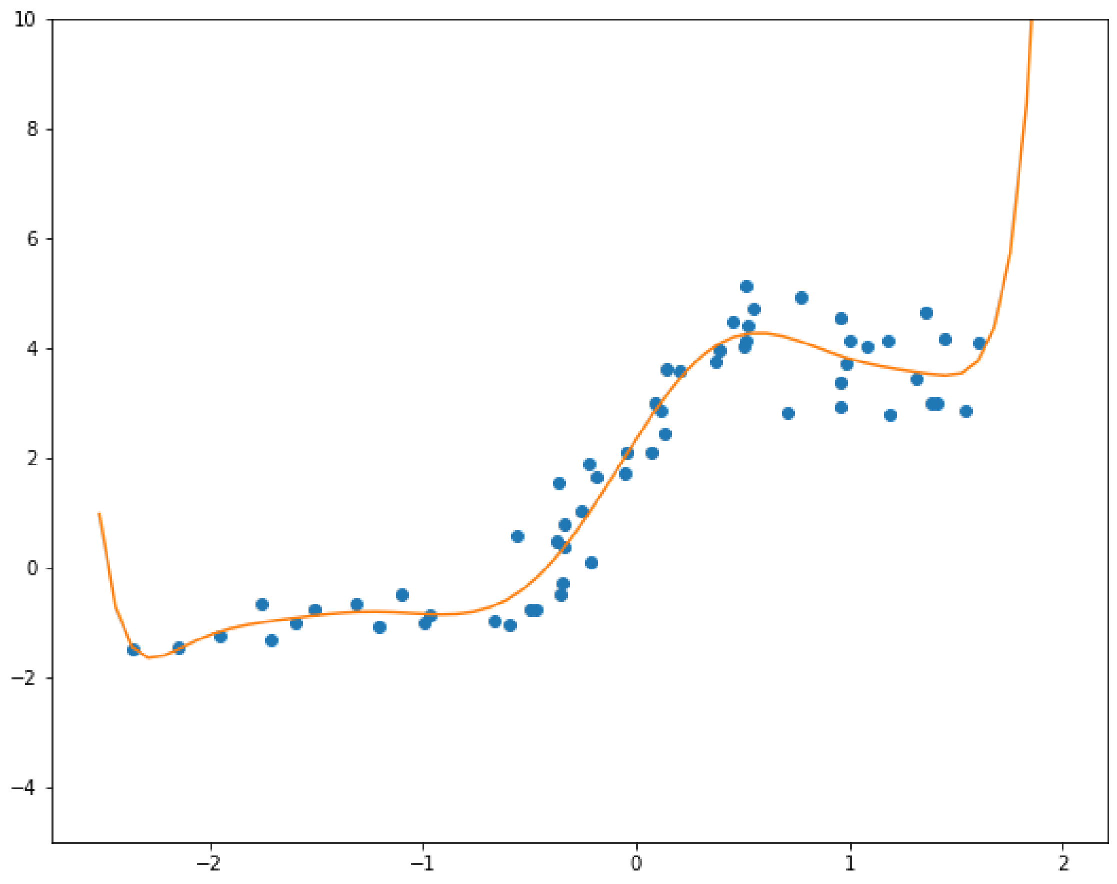
3b. d = 7

```
[133]: sum = 0
       for x in range(len(Ytr)):
           sum += pow((Ytr[x] - Yhat[x]), 2)
       error = sum/len(Ytr)
       print("training error =", error)

       sum2 = 0
       for x in range(len(Yte)):
           sum2 += pow((Yte[x] - Yhat2[x]), 2)
       error2 = sum2/len(Yte)
       print("testing error =", error2)
       Xtraining.append(error)
       Xtesting.append(error2)
```

```
training error = [0.31563467]
testing error = [0.65022461]
```

3a. d = 10

[134]:
```
# Create polynomial features up to "degree"; don't create constant feature
# (the linear regression learner will add the constant feature automatically)
XtrP = ml.transforms.fpoly(Xtr, 10, 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,10,False), params)

xs = np.linspace(0,10,60)
xs = xs[:,np.newaxis]
xsP,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,10,False), params)

new = lr.predict(xsP)

Yhat = lr.predict(XtrP)
Yhat2 = lr.predict(XteP)

fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
ax.plot(XtrP[:,0], Ytr, 'o') # Plot polynomial regression of desired degree
ax.plot(xsP[:,0], new)
ax.set_ylim(-5, 10) # Set the minimum and maximum limits
plt.show()
```



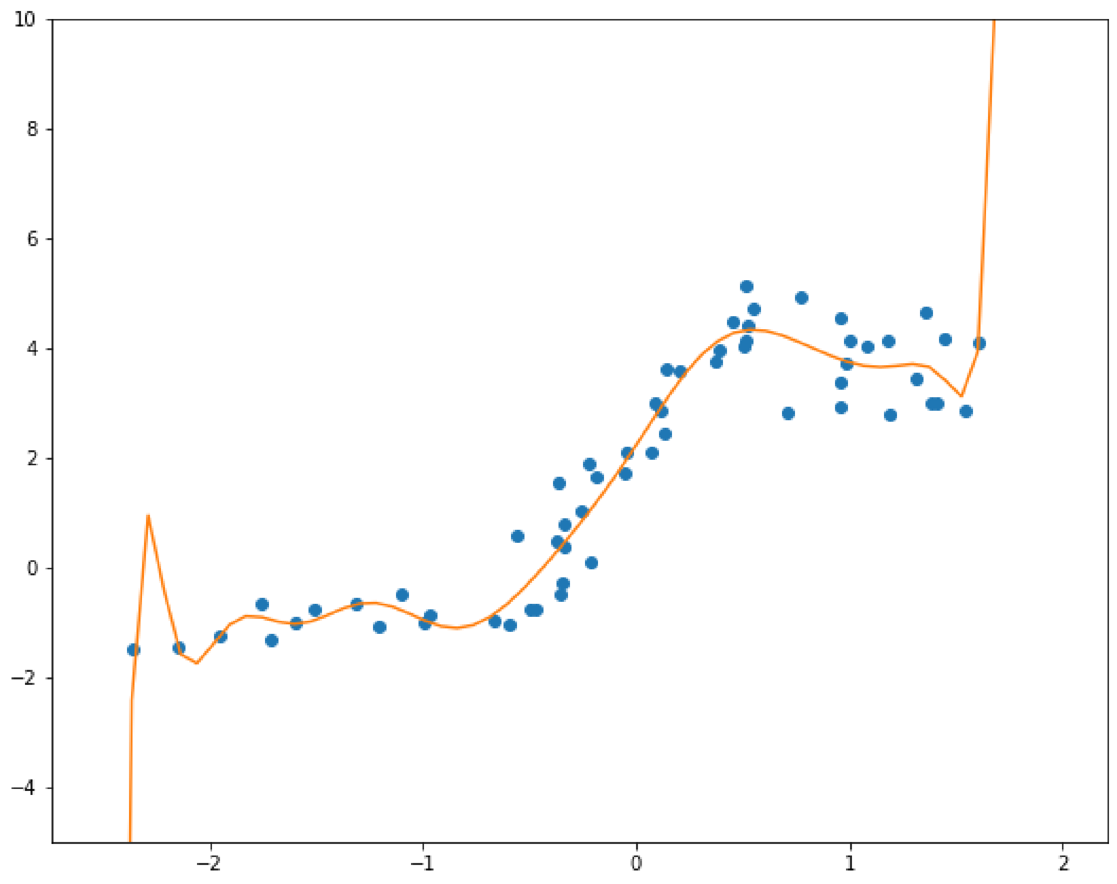
3b. d = 10

```
[135]: sum = 0
       for x in range(len(Ytr)):
           sum += pow((Ytr[x] - Yhat[x]), 2)
       error = sum/len(Ytr)
       print("training error =", error)

       sum2 = 0
       for x in range(len(Yte)):
           sum2 += pow((Yte[x] - Yhat2[x]), 2)
       error2 = sum2/len(Yte)
       print("testing error =", error2)
       Xtraining.append(error)
       Xtesting.append(error2)
```

```
training error = [0.29894798]
testing error = [0.60906007]
```

3a. d = 15

[136]:
```python
# Create polynomial features up to "degree"; don't create constant feature
# (the linear regression learner will add the constant feature automatically)
XtrP = ml.transforms.fpoly(Xtr, 15, 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,15,False), params)

xs = np.linspace(0,10,60)
xs = xs[:,np.newaxis]
xsP,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,15,False), params)

new = lr.predict(xsP)

Yhat = lr.predict(XtrP)
Yhat2 = lr.predict(XteP)

fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
ax.plot(XtrP[:,0], Ytr, 'o') # Plot polynomial regression of desired degree
ax.plot(xsP[:,0], new)
ax.set_ylim(-5, 10) # Set the minimum and maximum limits
plt.show()
```



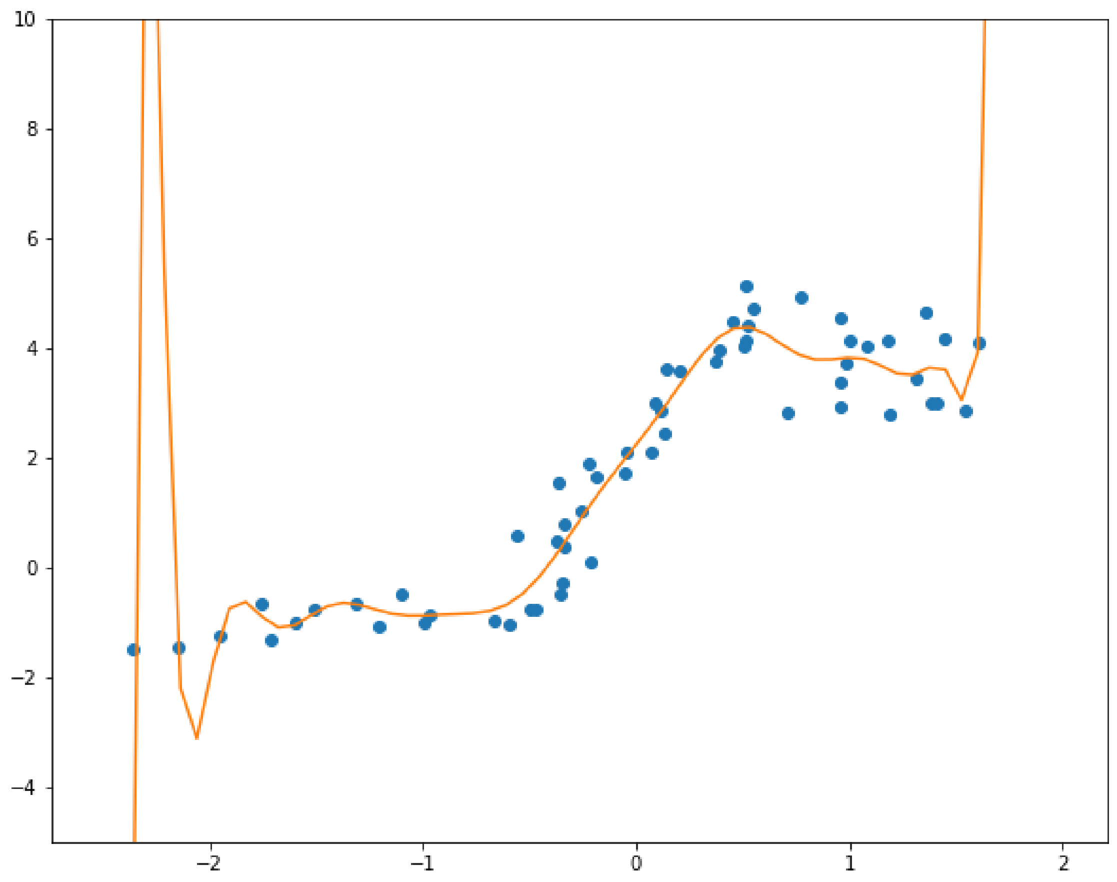
3b. d = 15

```
[137]: sum = 0
       for x in range(len(Ytr)):
           sum += pow((Ytr[x] - Yhat[x]), 2)
       error = sum/len(Ytr)
       print("training error =", error)

       sum2 = 0
       for x in range(len(Yte)):
           sum2 += pow((Yte[x] - Yhat2[x]), 2)
       error2 = sum2/len(Yte)
       print("testing error =", error2)
       Xtraining.append(error)
       Xtesting.append(error2)
```

```
training error = [0.28817931]
testing error = [7.86335909]
```

3a. d = 18

[138]:
```
# Create polynomial features up to "degree"; don't create constant feature
# (the linear regression learner will add the constant feature automatically)
XtrP = ml.transforms.fpoly(Xtr, 18, 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

#print(Xtr)

# Now, apply the same polynomial expansion & scaling transformation to Xtest:
XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte,18,False), params)

xs = np.linspace(0,10,60)
xs = xs[:,np.newaxis]
xsP,_ = ml.transforms.rescale( ml.transforms.fpoly(xs,18,False), params)

new = lr.predict(xsP)

Yhat = lr.predict(XtrP)
Yhat2 = lr.predict(XteP)

fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
ax.plot(XtrP[:,0], Ytr, 'o') # Plot polynomial regression of desired degree
ax.plot(xsP[:,0], new)
ax.set_ylim(-5, 10) # Set the minimum and maximum limits
plt.show()
```



3b. d = 18

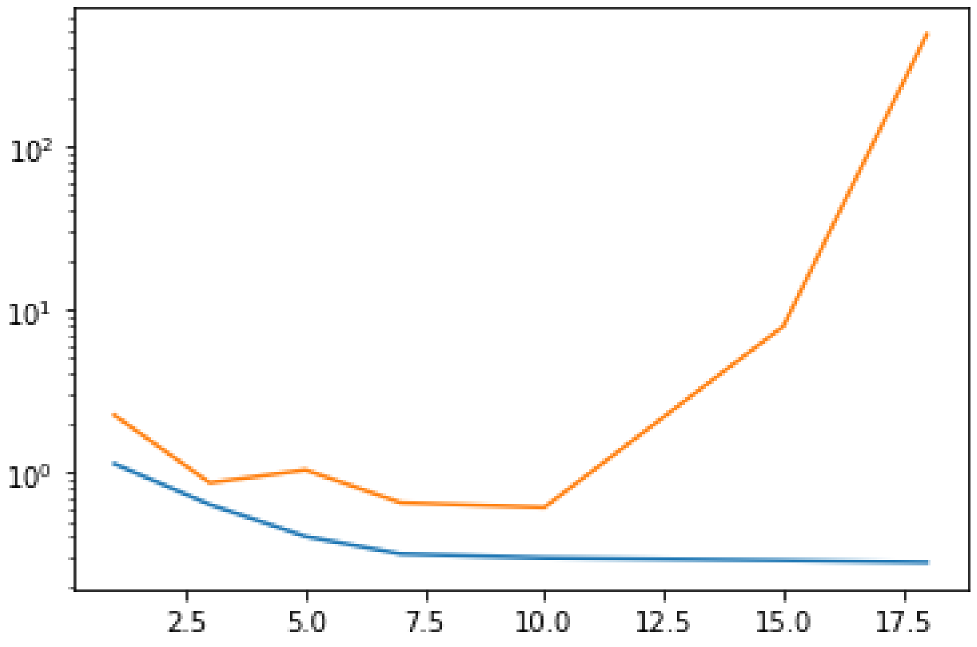
[139]:
```
sum = 0
for x in range(len(Ytr)):
    sum += pow((Ytr[x] - Yhat[x]), 2)
error = sum/len(Ytr)
print("training error =", error)

sum2 = 0
for x in range(len(Yte)):
    sum2 += pow((Yte[x] - Yhat2[x]), 2)
error2 = sum2/len(Yte)
print("testing error =", error2)
Xtraining.append(error)
Xtesting.append(error2)
```

```
training error = [0.28048505]
testing error = [482.28032737]
```

3b.

[140]:
```
Y = [1,3,5,7,10,15,18]
plt.semilogy(Y, Xtraining)
plt.semilogy(Y, Xtesting)
```

[140]:
```
[<matplotlib.lines.Line2D at 0x11a4ccdf0>]
```



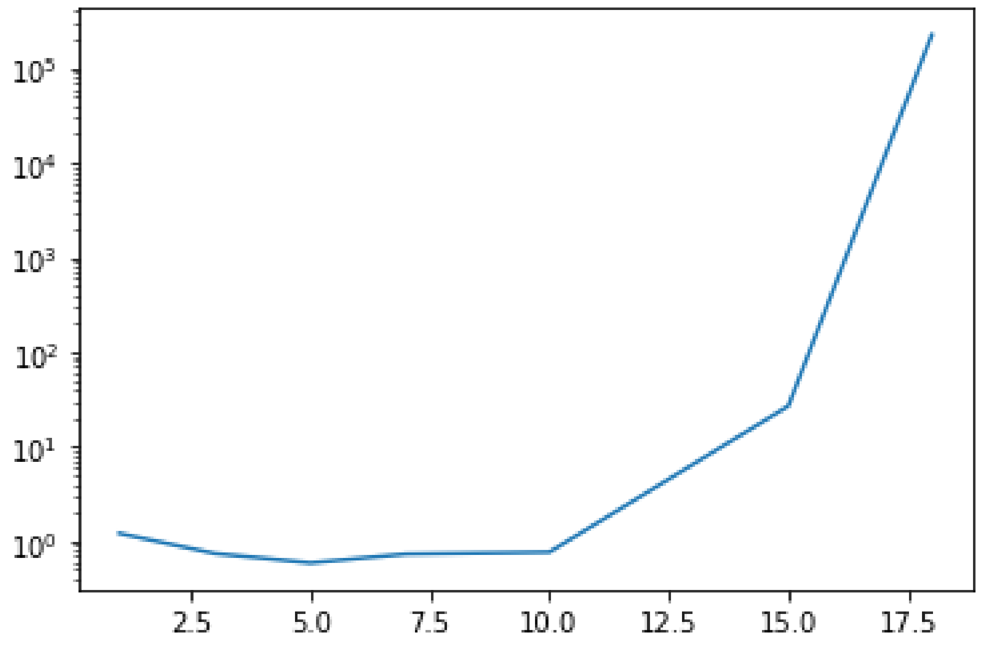
3c. I would recommend using polynomial degree 10 to model the data.

[141]:
```
def crossValidation(degree, folds):
    J = [0] * folds
    XtrP = ml.transforms.fpoly(Xtr, degree, bias=False)
    XtrP,params = ml.transforms.rescale(XtrP)
    for iFold in range(folds):
        Xti,Xvi,Yti,Yvi = ml.crossValidate(XtrP,Ytr,folds,iFold) # use ith␣
↪block as validation
        learner = ml.linear.linearRegress(Xti,Yti) # TODO: train on Xti, Yti,␣
↪the data for this fold
        Yhat = learner.predict(Xvi)
        sum = 0
        for i in range(len(Yvi)):
            sum += pow((Yvi[i] - Yhat[i]), 2)
        J[iFold] =  sum/len(Yvi)# TODO: now compute the MSE on Xvi, Yvi and␣
↪save it
    # the overall estimated validation error is the average of the error on␣
↪each fold
    return np.mean(J)
```

4a.

[142]:
```
Y = [1,3,5,7,10,15,18]
X =␣
↪[crossValidation(1,5),crossValidation(3,5),crossValidation(5,5),crossValidation(7,5),
    crossValidation(10,5), crossValidation(15,5),crossValidation(18,5)]
plt.semilogy(Y, X)
```

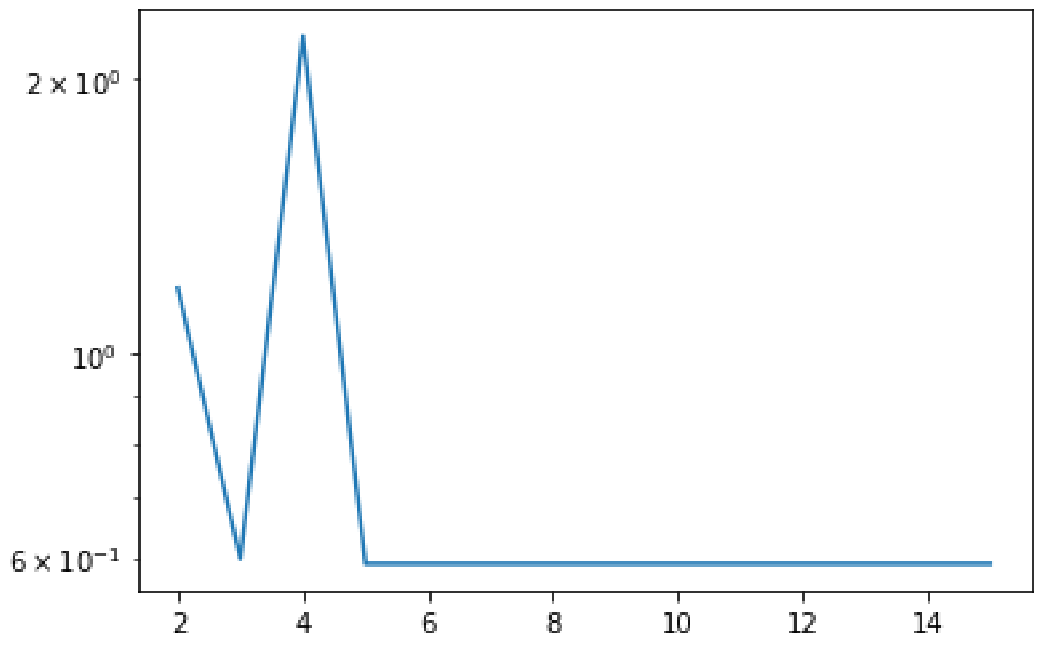
[142]:
```
[<matplotlib.lines.Line2D at 0x11ae425b0>]
```



4b. The MSE for five fold variation and the actual test data start low for lower degree polynomials, but the MSE increases much faster in five fold variation at higher degree polynomials.4c. I recommend a degree 5 polynomial.4d.

[143]:
```
Y = [2, 3, 4, 5, 6, 10, 12, 15]
X = []
for x in Y:
    if(x <= 5):
        X.append(crossValidation(5,x))
    else:
        X.append(crossValidation(5,5))
plt.semilogy(Y, X)
print(X)
```

```
[1.1795458641317584, 0.5984555010978755, 2.2195261560646546, 0.5910703726407058,
0.5910703726407058, 0.5910703726407058, 0.5910703726407058, 0.5910703726407058]
```



Training data increases as n fold increases since the validation data you choose becomes smaller for each fold. As training data increases, generally speaking, your error will decrease since you have more points to learn from. However, at 4-fold, the error increases rather than decreasing, which means there must have been outlier data in one of the folds.

[ ]:

```

```

Statement of Collaboration: I discussed the structure of the random test data in question 3 and how the cross validation function in 4a (as well as general cross validation) works with Luke Yi.