Assignment 1B

Delaney Gomen

My process

I ended up using M=8 and k=0.05100. How did I reach this result? I trained the model for varying levels of k and M (like told in the directions), but instead of using the graphs where we had to calculate the error for each model iteration with the testing data, I used the results I found from the *training data* by calculating the error on the training sets. I did not want to choose my parameters based off of the performance of the models on the testing data, even though that's what we were asked to plot. This was inspired by a comment Dr. Zare made on Canvas: "The"best" parameters may be different for training and testing data. However, it practice, you would only have training data to select parameters (using a cross-validation style setup). So, it is a little like you are cheating if you use the test data to select parameter settings. The test data should only be to evaluate/test the model."

Questions

1) Does the Huber M-Estimator outperform (in terms of error between predicted and desired) the standard Least Squares solution on the provided training and testing data? Why or why not?

Yes, I believe that the Huber M-Estimator outperforms the standard Least Squares solution on both the testing and training data. In the graph below, we can see that the IRLS model fits the test data better than the LS model-but very, very slightly. The main outliers in the data affect the Huber M-Estimator far less than the Least Squares (you can see the LS has a higher "peak" influenced by the outliers directly above).

```
p1=plt.plot(x3, t3, 'grey', marker='.', linewidth=0, alpha=0.5)
p2=plt.plot(x3, yIRLStest, 'orange',linewidth=2)
p3=plt.plot(x3, yLStest, 'blue', linewidth=2)

plt.legend((p2[0], p3[0]), ('IRLS', 'LS'))
```

```
<matplotlib.legend.Legend at 0x1a29586e10>
```

2) Do any of the generated plots show indication of overfitting in any of your results? Why or why not?

Yes. Especially when M approaches 10 or more, we see very strange patterns at the end ranges of the model, where the model zig-zags to perfectly fit the training data. This is expected with higher-order polynomials, as the weights get better tuned to the random variability of the model.

3) What is the role of the parameter k and how does the choice of k effect Huber M-Estimator results?

The role of k is to check for outliers that affect the model. Because the model is fitted onto the data before the errors are tested against k, it is important to make sure that your original model is not severly overfitted, or else the errors will be small and have no effect on the weights. However, this is why the choice of k also effects the estimator results. Even if your model is severly overfitted to begin with, if your k is very very small, it will still find outliers and change their weighted coefficient. So, the smaller the k, the more sensitive the algorithm is to outliers. As k approaches 0, the weights on outliers get smaller.

4) Using the Huber M-estimator amounts to a weighted linear regression problem with the weights defined as B above. Provide a derivation for why these are the correct weight values (or prove otherwise). In other words, prove (or disprove) that the equation for bii given above is correct. Show all of your work.

See pages attached to this file.

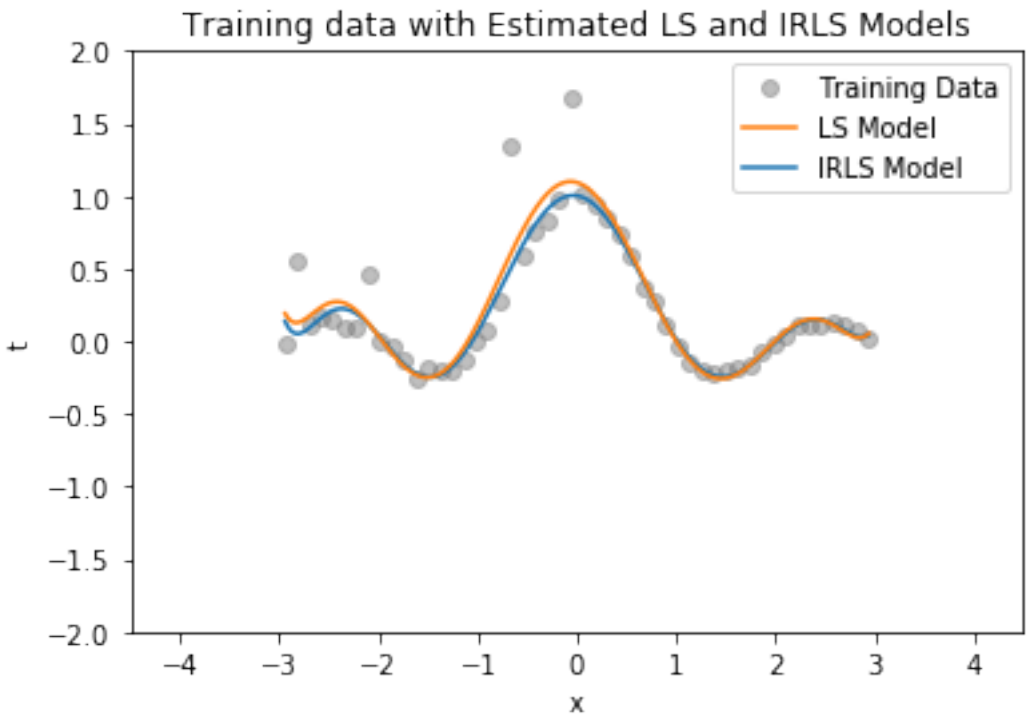

Figure 1: png

```
plt.title('Mean Absolute Value on Test Data for Varying M')

plt.xlabel('M')
plt.ylabel('Mean Absolute Value for (y_i - t_i)')

f3 = plt.figure(3)

p1 = plt.plot(numk, errors_IRLSk)

plt.title('Mean Absolute Value on Test Data for Varying k')

plt.xlabel('k')
plt.ylabel('Mean Absolute Value for (y_i - t_i)')

plt.show()

0.051000000000000004
0.08901811671353252
9
0.051000000000000004
0.08148547319412405
0.08681844369948732
```

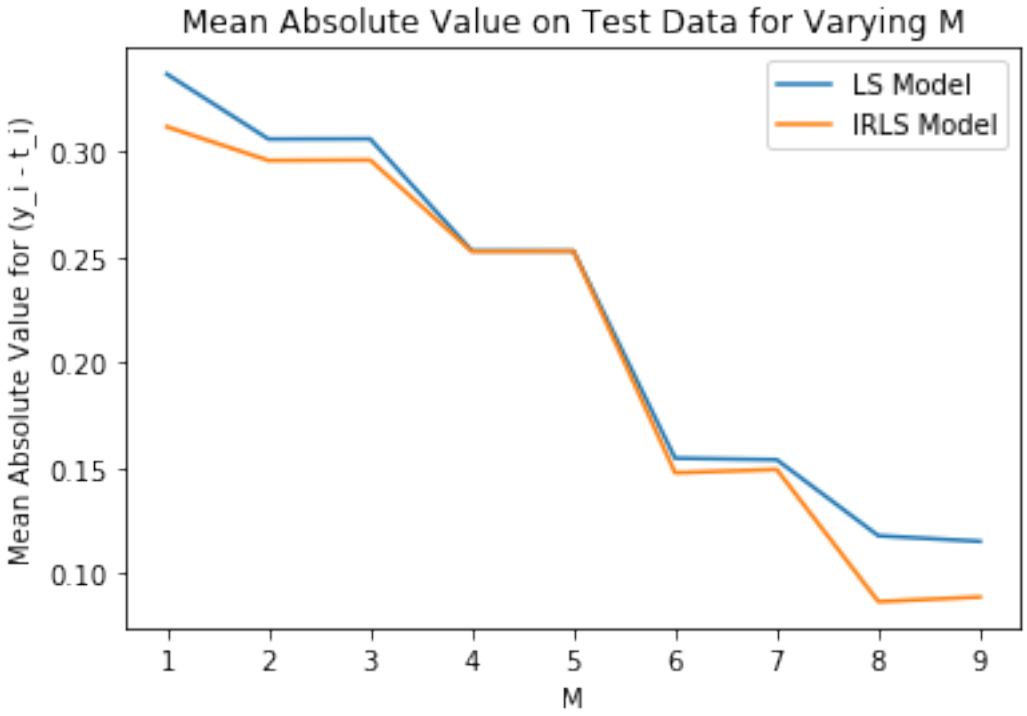

Figure 2: png

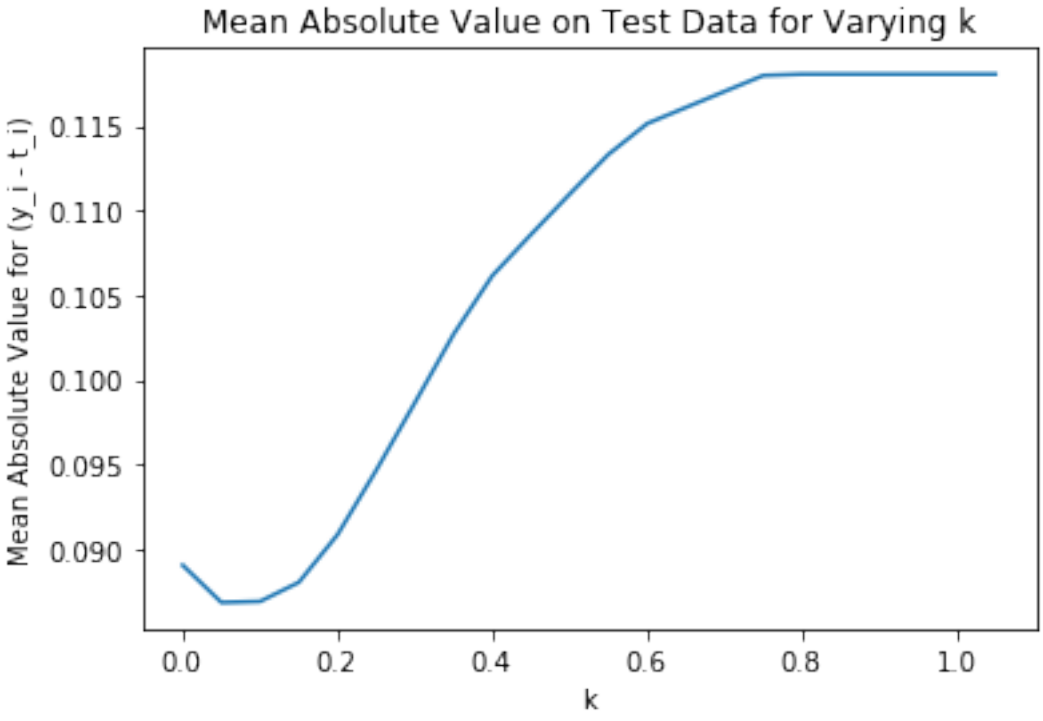

Figure 3: png

We have

$$J(w) = \begin{cases} \sum_{n=1}^{N} \frac{1}{2} (t_n - y_n)^2 \; ; & |t_n - y_n| \le k \\ \sum_{n=1}^{N} k |t_n - y_n| - \frac{1}{2} k^2 \; ; & |t_n - y_n| > k \end{cases}$$

$$= \begin{cases} \sum_{n=1}^{N} \frac{1}{2} (t_n - \sum_{j=0}^{M} w_j x_n^j)^2 \; ; & |t_n - y_n| \le k \\ \sum_{n=1}^{N} k \left| t_n - \sum_{j=0}^{M} w_j x_n^j \right| - \frac{1}{2} k^2 \; ; & |t_n - y_n| > k \end{cases}$$

And

$$\frac{\partial J(w)}{\partial w} = \left[\frac{\partial J(w)}{\partial w_0}, \cdots \frac{\partial J(w)}{\partial w_M} \right]^T$$

Where

$$\frac{\partial J(w)}{\partial w_j} = \begin{cases} \sum_{n=1}^{N} \left(t_n - \sum_{j=0}^{M} w_j x_n^j \right) x_n^j & |t_n - y_n| \le k \\ * \sum_{n=1}^{N} k \dfrac{\left[\left(t_n - \sum_{j=0}^{M} w_j x_n^j \right) x_n^j \right]}{\left| t_n - \sum_{j=0}^{M} w_j x_n^j \right|} & |t_n - y_n| > k \end{cases}$$

* Which was found by the derivation rule $\frac{\partial}{\partial x}|u| = \frac{u}{|u|}\frac{\partial u}{\partial x}$

* Tells us that the new optimization matrix X^T will consist of the same entries of the original Least-squares X^T, but when $|t_n - y_n| > k$, the entire column will have a coefficient of $\frac{k}{|t_n - y_n|}$.

Say X^T_{LS} is composed w/ the least squares function. Then:

$$X^T = \begin{bmatrix} 1 & \cdots & 1 \\ x_1 & \ddots & x_N \\ \vdots & \ddots & \vdots \\ x_1^m & \cdots & x_N^m \end{bmatrix} = \begin{bmatrix} v_1 & v_2 \cdots & v_n \end{bmatrix}$$

If $|t_n - y_n| > k$ for any $n = 1, 2, \ldots N$, then

$v_n = \frac{k}{|t_n - y_n|} v_n$. So:

$$X^T_B = \begin{cases} v_n & \text{for } |t_n - y_n| \leq k \\ \frac{k}{|t_n - y_n|} v_n & \text{for } |t_n - y_n| > k \end{cases}$$

If we construct a diagonal matrix B, then for $n = 1, 2, \dots, N$ where $|t_n - y_n| > k$, we have $b_{nn} = \frac{k}{|t_n - y_n|}$. Because

$e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ $e_m = \begin{pmatrix} 0 \\ \vdots \\ 1 \end{pmatrix}$

Where b_n come from, $\begin{bmatrix} b_1 & \dots & b_N \end{bmatrix} = B$

$X^T B$ where $B = \begin{cases} b_n = e_n \text{ (identity column vector)} & \text{for } |t_n - y_n| \le k \\ b_n = \frac{k}{|t_n - y_n|} e_n & \text{for } |t_n - y_n| > k \end{cases}$

Is equal to X^T_B. So, the coefficients b_{ii} are correct.

$$X^T_B = \begin{bmatrix} \frac{k}{|t_n - y_n|}(1) & \dots & 1 \\ \vdots & & \vdots \\ \frac{k}{|t_n - y_n|}(x_1^m) & \dots & x_N^m \end{bmatrix} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & & \vdots \\ x_1^m & & x_N^m \end{bmatrix} \begin{bmatrix} \frac{k}{|t_n - y_n|} & \dots & 0 \\ 0 & \ddots & 0 \\ \vdots & & \\ 0 & \dots & 1 \end{bmatrix}$$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

plt.close('all') #close any open plots

"""
================================================================================
================================================================================
============================ Question 1 ========================================
================================================================================
================================================================================
"""
""" ======================  Function definitions ========================== """

def plotData(x1,t1,x2=None,t2=None,x3=None,t3=None,legend=[]):
    '''plotData(x1,t1,x2,t2,x3=None,t3=None,legend=[]): Generate a plot of the
       training data, the true function, and the estimated function'''
    p1 = plt.plot(x1, t1, 'bo') #plot training data
    if(x2 is not None):
        p2 = plt.plot(x2, t2, 'g') #plot true value
    if(x3 is not None):
        p3 = plt.plot(x3, t3, 'r') #plot training data

    #add title, legend and axes labels
    plt.ylabel('t') #label x and y axes
    plt.xlabel('x')

    if(x2 is None):
        plt.legend((p1[0]),legend)
    if(x3 is None):
        plt.legend((p1[0],p2[0]),legend)
    else:
        plt.legend((p1[0],p2[0],p3[0]),legend)
    return

def fitdataLS(x,t,M):
    '''fitdataLS(x,t,M): Fit a polynomial of order M to the data (x,t) using LS'''
    #This needs to be filled in
    X = np.array([x**m for m in range(M+1)]).T
    w = np.linalg.inv(X.T@X)@X.T@t

    return w

def fitdataIRLS(x,t,M,k):
    '''fitdataIRLS(x,t,M,k): Fit a polynomial of order M to the data (x,t) using IRLS'''
    #This needs to be filled in

    X = np.array([x**m for m in range(M+1)]).T
    w = fitdataLS(x, t, M)
    w_prev = np.zeros(len(w))
    y = X@w
```

```
    while all(abs(w - w_prev)) > 0.001:

        b_list = []

        for i in np.arange(len(x)):

            if abs(t[i] - y[i]) <= k:
                b_list += [1]

            elif abs(t[i] - y[i]) > k:
                b_list += [k / abs(t[i] - y[i])]


        B = np.diagflat(b_list)

        w_prev = w

        w = np.linalg.inv(X.T@B@X)@X.T@B@t

    return w


def modeltesting(testM, testk, M, k, xtrain, xtest, ttrain, ttest):

    errors_LS = []
    errors_IRLS = []
    errors_LStrain = []
    errors_IRLStrain = []
    errors_dict = {}

    if testM:

        for i in np.arange(1,M+1):

            Xtest = np.array([xtest**i for i in range(i+1)]).T
            Xtrain = np.array([xtrain**i for i in range(i+1)]).T

            wLS = fitdataLS(xtrain, ttrain, i)
            wIRLS = fitdataIRLS(xtrain, ttrain, i, k)

            yLStest = Xtest@wLS
            yIRLStest = Xtest@wIRLS

            yLStrain = Xtrain@wLS
            yIRLStrain = Xtrain@wIRLS

            errors_LS += [np.mean(abs(ttest - yLStest))]
            errors_IRLS += [np.mean(abs(ttest  - yIRLStest))]

            errors_LStrain += [np.mean(abs(ttrain - yLStrain))]
            errors_IRLStrain += [np.mean(abs(ttrain  - yIRLStrain))]

            errors_dict[str(np.mean(abs(ttrain  - yIRLStrain)))] = i
```

```
        return np.arange(1,M+1), errors_LS, errors_IRLS, errors_LStrain, errors_IRLStrain, errors_dict

    if testk:

        for i in np.arange(0.001, k+0.1, 0.05):

            Xtest = np.array([xtest**m for m in range(M+1)]).T
            Xtrain = np.array([xtrain**m for m in range(M+1)]).T

            wIRLS = fitdataIRLS(xtrain, ttrain, M, i)

            yIRLStest = Xtest@wIRLS
            yIRLStrain = Xtrain@wIRLS

            errors_IRLS += [np.mean(abs(ttest  - yIRLStest))]
            errors_IRLStrain += [np.mean(abs(ttrain  - yIRLStrain))]

            errors_dict[str(np.mean(abs(ttrain  - yIRLStrain)))] = i

        return np.arange(0.001, k+0.1, 0.05), errors_IRLS, errors_IRLStrain, errors_dict


""" ======================  Variable Declaration ========================== """
M = 5 #regression model order
k = 0.01 #Huber M-estimator tuning parameter

""" ======================  Load Training Data ======================= """
data_uniform = np.load('TrainData.npy')
x1 = data_uniform[:,0]
t1 = data_uniform[:,1]

""" ======================  Train the Model ============================== """
wLS = fitdataLS(x1,t1,M)
wIRLS = fitdataIRLS(x1,t1,M,k)


# In[384]:


""" ======================= Load Test Data  and Test the Model ========================== """

"""This is where you should load the testing data set. You shoud NOT re-train the model   """

data_test = np.load('TestData.npy')
x3 = data_test[:,0]
t3 = data_test[:,1]

numk, errors_IRLSk, errors_IRLSktrain, errors_dict = modeltesting(testM=False, testk=True, M=8, k=1,
                                            xtrain=x1, xtest=x3, ttrain=t1, ttest=t3)

numM, errors_LS, errors_IRLS, errors_LStrain, errors_IRLStrain, errors_dict_M = modeltesting(testM=True
                                            xtrain=x1, xtest=x3, ttrain=t1, ttest=t3)
```

```
M = errors_dict_M[str(min(errors_IRLStrain))] #regression model order
k = errors_dict[str(min(errors_IRLSktrain))] #Huber M-estimator tuning parameter
print(k)
Xtest = np.array([x3**M for M in range(M+1)]).T

# Using training data on best parameters
wIRLS = fitdataIRLS(x1, t1, M, k)
yIRLStest = Xtest@wIRLS

wLS = fitdataLS(x1, t1, M)
yLStest = Xtest@wLS

test_error = np.mean(abs(t3  - yIRLStest))

print(test_error)

""" ========================  Plot Results ============================== """

""" This is where you should create the plots requested """

print(errors_dict_M[min(errors_dict_M)])
print(errors_dict[min(errors_dict)])
print(min(errors_IRLSktrain))
print(min(errors_IRLS))

f1  = plt.figure(1)

plt.ylim(-2,2)
plt.xlim(-4.5,4.5)
plt.title('Training data with Estimated LS and IRLS Models')

wIRLStrain = fitdataIRLS(x1, t1, M, k)
wLStrain = fitdataLS(x1, t1, M)
xrange = np.arange(min(x1),max(x1),0.01)  #get equally spaced points in the xrange
Xtrain = np.array([xrange**m for m in range(wIRLStrain.size)]).T

p1 = plt.plot(x1, t1, 'grey', marker='o', linewidth=0, alpha=0.5)
p2 = plt.plot(xrange, Xtrain@wIRLStrain)
p3 = plt.plot(xrange, Xtrain@wLStrain)

plt.legend((p1[0], p3[0],p2[0]),('Training Data', 'LS Model', 'IRLS Model'))

plt.xlabel('x')
plt.ylabel('t')

plt.draw()


f2 = plt.figure(2)

p1 = plt.plot(numM, errors_LS)
p2 = plt.plot(numM, errors_IRLS)

plt.legend((p1[0], p2[0]), ('LS Model', 'IRLS Model'))
```