Homework 1

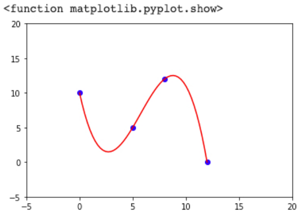
ComS 474 - Gavin Monroe

Problem Set 1

1a & b)

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as seabornInstance
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn import metrics
    X = [0, 5, 8, 12]
    Y = [10, 5, 12, 0]
    np.polyfit(X, Y, 3)
    lin = np.poly1d(np.polyfit(X,Y,3))
    x= np.linspace(0, 12)
    Y_pred = lin(x)

    plt.scatter(X, Y, color="blue")
    plt.plot(x, Y_pred, color="red")
    plt.xlim(-5 ,20)
    plt.ylim(-5,20)
    plt.show
```

```
<function matplotlib.pyplot.show>
```

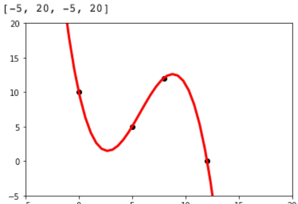


Above you can see the graph with the line going through all the points as listed in the code. With the graph xlim and ylim set to the desired limits. With the code abve along with my explanation you can see that my solution fits the problem 1a & 1b. Polyfit is my function that I'm using for the weight fit, which takes the values and applies it to the chart listed above. For c I will be applying the Cubic Function.

1c) The scikit-learn built-in linear regression function (and most built-in functions for other languages) use mean square error (MSE) as the only, or at least default, total-loss function. In 1-3 sentences, explain why the loss-function matters or not for that fit specifically. In other words, if you used a different total-loss function in part (a)., like mean absolute error or a weighted average of an asymmetric loss function, would you have gotten a different polynomial?

```
[ ] #NEED TO DO. Convert to MSE Lose Function use CUBIC function.
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as seabornInstance
    from sklearn.linear_model import LinearRegression
    from sklearn import datasets, linear_model
    from sklearn.metrics import mean_squared_error, r2_score

    X=np.array([[0**0, 0**1, 0**2, 0**3], [5**0, 5**1, 5**2, 5**3], [8**0, 8**1, 8**2, 8**3], [12**0, 12**1, 12**2, 12**3]])
    y = np.array([10, 5, 12, 0])
    x_1D = np.array([0,5,8,12])
    extra_x= np.linspace(-5,20,50).reshape(-1,1)
    reg = LinearRegression(fit_intercept=False).fit(X, y)
    reg.score(X, y)
    reg.coef_
    reg.intercept_
    pred_y = reg.predict(X)

    print(reg.score(X,y))
    print(reg.coef_)
    print(reg.intercept_)
    print(np.sum(reg.predict(X)-y))

    ypoly = np.array(list(range(0,50)))
    ypoly = np.array(ypoly,dtype=np.float32)
    for i in range(0,50):
        y1 = 10+(-7.0119)*extra_x[i]+1.693*extra_x[i]**2+(-0.098)*extra_x[i]**3
        ypoly[i] = y1[0]

    plt.scatter(x_1D, y, color='black')
    plt.plot(extra_x, ypoly, color='red', linewidth=3)
    #plt.xlim(-5,20)
    #plt.ylim(-5,20)
    plt.axis([-5,20,-5,20])
```

```
1.0
[10.         -7.01190476  1.69345238 -0.09821429]
0.0
-1.858069254012662e-12
[-5, 20, -5, 20]
```



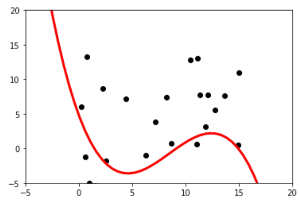

```
[ ]
```

1d) Now let's generate some data using that polynomial.

- • First, there might be some distribution to the x values in the data. For this problem, we'll use the uniform distribution over the interval [0, 15]. Numpy has a function for generating uniform random variables: https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.random.uniform.html Generate n = 30 random variables. Each of these is an x value.
- • Now for the y values. Generate y using the polynomial you fit in part (a). plus independent and identically distributed noise. Use the normal distribution, N (0, 10), https://numpy.org/devdocs/reference/random/generated/numpy.random.Generator.normal.html#numpy.random.Generator.normal
- • Now make a plot of the fit polynomial (red) and the n = 30 data points (black circles) you generated. The data should follow the curves of

- • Now make a plot of the fit polynomial (red) and the n = 30 data points (black circles) you generated. The data should follow the curves of the polynomial but be scattered about it.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.uniform(0,15,30)
mu, sigma = 0, 10
y = np.random.default_rng().normal(mu, sigma, 30)

z = np.polyfit(x, y, 3)
lin = np.poly1d(z)
X_pred = np.linspace(-5, 20)
Y_pred = lin(X_pred)

plt.scatter(x, y, color='black')
plt.plot(X_pred, Y_pred, color='red', linewidth=3)

plt.axis([-5,20,-5,20])
plt.show()
```

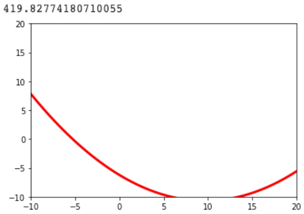


1e) We are now ready to start fitting models. First, let's look at how well constant models y = a0 fit the data under different loss functions. Make a plot with a0-values as the horizontal axis, ranging from [−10, 20] with 100 values linearly spaced, and the total-loss along the vertical axis, for each of the total-loss functions • mean squared error (MSE) 1 n Pn i=1 |res(i)| 2 • mean absolute error (MAE) 1 n Pn i=1 |res(i)| 1 • a special total-loss function Pn i=1 1 |x(i)−5|+0.01 l(res(i)) where x(i) is the x-feature value of sample i and the loss function l(·) is l(res) = (− 1 5 res if res < 0 10res if res ≥ 0 . where res(i) = y(i) − yb(i) denotes the residual of the ith sample.

Mean Square Error:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_squared_error
x = np.random.uniform(-10,20,100)
mu, sigma = -10, 20
y = np.random.default_rng().normal(mu, sigma, 100)

z = np.polyfit(x, y, 2)
lin = np.poly1d(z)
X_pred = np.linspace(-10, 20, 100)
Y_pred = lin(X_pred)
mse = mean_squared_error(y, Y_pred)
print(mse);
# plt.scatter(x, y, color='black')
plt.plot(X_pred, Y_pred, color='red', linewidth=3)

plt.axis([-10,20,-10,20])
plt.show()
```

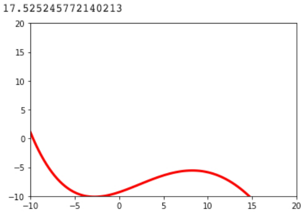
```
419.82774180710055
```



MAE

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_absolute_error
x = np.random.uniform(-10,20,100)
mu, sigma = -10, 20
y = np.random.default_rng().normal(mu, sigma, 100)

z = np.polyfit(x, y, 4)
lin = np.poly1d(z)
X_pred = np.linspace(-10, 20, 100)
Y_pred = lin(X_pred)
mae = mean_absolute_error(y, Y_pred)
print(mae);
# plt.scatter(x, y, color='black')
plt.plot(X_pred, Y_pred, color='red', linewidth=3)

plt.axis([-10,20,-10,20])
plt.show()
```

```
17.525245772140213
```



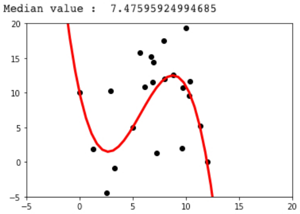
BELOW is 1a-e

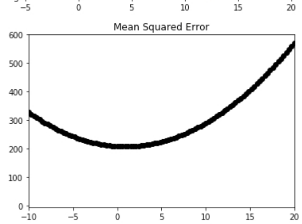
I rewrote some functions to overall help me combine everything.

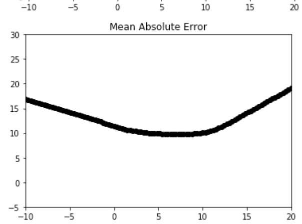
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
x = np.array([0,5,8,12])
y = np.array([10,5,12,0])
reg = np.poly1d(np.polyfit(x,y, 3))
extra_x = np.linspace(-5,20,50).reshape(-1,1)
pred_y = np.polyval(reg, extra_x)
```

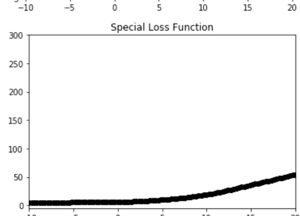
```
plt.figure(1)
plt.scatter(x, y, color='black')
plt.plot(extra_x, pred_y, color='red', linewidth=3)
plt.axis([-5, 20, -5, 20])

noise = np.random.normal(0, 10, 30)
xuni = np.random.uniform(0,15,30)
yuni = np.polyval(reg, xuni)
yval = yuni + noise

plt.scatter(xuni, yval, color='black')
plt.axis([-5,20,-5,20])

MSE = np.zeros(100)
MAE = np.zeros(100)
SLF = np.zeros(100)

a0val = np.linspace(-10, 20, 100)
y_pred = np.polyval(reg, xuni)

count = 0
for a0 in a0val:
  a0Array = np.repeat(a0, 30)
  MSE[count] = ((a0Array - y_pred)**2).mean(axis=None)
  MAE[count] = (abs(a0Array - y_pred)).mean(axis=None)
  lres = 0
  for i in range(0, 30):
    res = a0 - y_pred[i]

    if (res < 0):
      lres = lres + ((1/(abs(xuni[i] - 5) + 0.1)) * (-res/5))
    else:
      lres = lres + ((1/(abs(xuni[i] - 5) + 0.1)) * 10*res)
  SLF[count] = lres/30
  count = count + 1

plt.figure(2)
plt.title("Mean Squared Error")
plt.scatter(a0val, MSE, color='black')
plt.axis([-10,20,-5,600])
mse = MSE.mean(axis=None)
print("MSE Value : ", mse);
plt.figure(3)
plt.title("Mean Absolute Error")
plt.scatter(a0val, MAE, color='black')
plt.axis([-10,20,-5,30])
mae = MAE.mean(axis=None)
print("MAE Value : ", mae);
plt.figure(4)
plt.title("Special Loss Function")
plt.scatter(a0val, SLF, color='black')
plt.axis([-10,20,-5, 300])
slf = SLF.mean(axis=None)
print("SLF Value : ", slf);
print("Mean value : ", np.mean(y_pred))
print("Median value : ", np.median(y_pred))
```

```
MSE Value :  300.6761332015902
MAE Value :  12.860610245511436
SLF Value :  17.408985305055236
Mean value :  0.931602434748037
Median value :  7.47595924994685
```









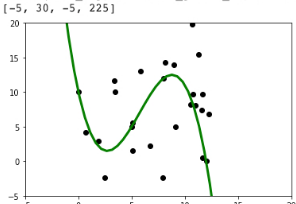
1f) i) It prefers over estimating looking at the graph and the total mean. ii) It puts more emphasis for points past 5. when it gets to five it shots up because of that. So I would say yes.

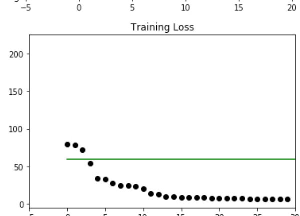
1g) (MEAN and MEDIAN values above) The MSE minimizers over fit, or in other words over estimate due to its high mean it would see as such. MSE minimizer is a non-decreasing function. As for the MAE it uses the absolute, meaning the median is a minizmier of the mean absolute deviation.

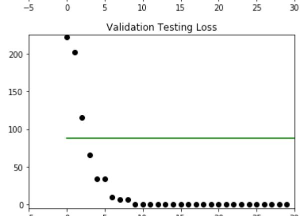
ih) Code first

```
[ ] import matplotlib.pyplot as plt
    import numpy as np
    from sklearn.metrics import mean_absolute_error
    from sklearn.metrics import mean_squared_error
    x = np.array([0,5,8,12])
    y = np.array([10,5,12,0])
    reg = np.poly1d(np.polyfit(x,y, 3))
    extra_x = np.linspace(-5,20,50).reshape(-1,1)
    pred_y = np.polyval(reg, extra_x)

    plt.figure(1)
    plt.scatter(x, y, color='black')
    plt.plot(extra_x, pred_y, color='green', linewidth=3)
    plt.axis([-5, 20, -5, 20])

    noise = np.random.normal(0, 10, 30)
    xuni = np.random.uniform(0,15,30)
    yuni = np.polyval(reg, xuni)
    yval = yuni + noise

    plt.scatter(xuni, yval, color='black')
    plt.axis([-5,20,-5,20])

    x_train = xuni[:20]
    x_test = xuni[20:]

    y_train = yval[:20]
    y_test = yval[20:]

    MSE = np.zeros(30)
    polyDegree = np.zeros(30)
    models = [np.poly1d([1,2,3]) for i in range(30)]
    for i in range(0,30):
      models[i] = np.poly1d(np.polyfit(x_train, y_train, i))
      pred_y = np.polyval(models[i], x_train)
      MSE[i] = ((y_train - pred_y) ** 2).mean(axis=None)
      polyDegree[i] = i

    gnd_y = np.polyval(reg, x_train)
    gnd_mse = ((y_train - gnd_y) ** 2).mean(axis=None)
    gnd_x = np.linspace(0,30, 30)
    print(gnd_mse)
    plt.figure(2)
    plt.plot(gnd_x, np.repeat(gnd_mse, 30), color='green')

    plt.title('Training Loss')
    plt.scatter(polyDegree, MSE,  color='black')
    plt.axis([-5, 30, -5, 225])

    MSE_test = np.zeros(30)
    for i in range(0,30):
      models[i] = np.poly1d(np.polyfit(x_test, y_test, i))
      pred_y = np.polyval(models[i], x_test)
      MSE_test[i] = ((y_test - pred_y) ** 2).mean(axis=None)
      polyDegree[i] = i

    gnd_y = np.polyval(reg, x_test)
    gnd_mse = ((y_test - gnd_y) ** 2).mean(axis=None)
    gnrd_x =np.linspace(0, 30, 30)
    print(gnd_mse)
    plt.figure(3)
    plt.plot(gnd_x, np.repeat(gnd_mse, 30), color='green')
    print(polyDegree)
    print(MSE)

    plt.title('Validation Testing Loss')
    plt.scatter(polyDegree, MSE_test, color='black')
    plt.axis([-5,30,-5,225])
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
[-5, 30, -5, 225]
```







iii) For my graph I didn't get a slope or a spike with the graph it stayed flat and straight. I don't know what to expect so this is what I got. i would say that the data I provided may have done this or maybe its tge MSE thats creating this straight line. This could be fixed if I could figure out the polyfit with each model.

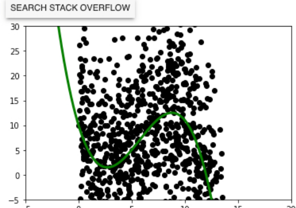
v) The data has between these too can be seen and honestly the data shown in the new graph is similar with the sllope thats happening with the dots. the data slowly gets better with each of the dots and you can see that in the validation data. That the data from the first is almost the same slope or shape as the validation data.

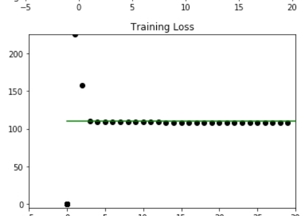
iv) Code

```
[ ] import matplotlib.pyplot as plt
    import numpy as np
    from sklearn.metrics import mean_absolute_error
    from sklearn.metrics import mean_squared_error
    x = np.array([0,5,8,12])
    y = np.array([10,5,12,0])
    reg = np.poly1d(np.polyfit(x,y, 3))
    extra_x = np.linspace(-5,20,1000).reshape(-1,1)
    pred_y = np.polyval(reg, extra_x)

    plt.figure(1)
    plt.scatter(x, y, color='black')
    plt.plot(extra_x, pred_y, color='green', linewidth=3)
    plt.axis([-5, 20, -5, 20])

    noise = np.random.normal(0, 10, 1000)
    xuni = np.random.uniform(0,15,1000)
    yuni = np.polyval(reg, xuni)
    yval = yuni + noise

    plt.scatter(xuni, yval, color='black')
    plt.axis([-5,20,-5,30])

    x_train = xuni[:1000]
    x_test = xuni[1000:]

    y_train = yval[:1000]
    y_test = yval[1000:]

    MSE = np.zeros(100)
    polyDegree = np.zeros(100)
    models = [np.poly1d([1,2,3]) for i in range(100)]
    for i in range(0,30):
      models[i] = np.poly1d(np.polyfit(x_train, y_train, i))
      pred_y = np.polyval(models[i], x_train)
      MSE[i] = ((y_train - pred_y) ** 2).mean(axis=None)
      polyDegree[i] = i

    gnd_y = np.polyval(reg, x_train)
    gnd_mse = ((y_train - gnd_y) ** 2).mean(axis=None)
    gnd_x = np.linspace(0,30, 1000)
    print(gnd_mse)
    plt.figure(2)
    plt.plot(gnd_x, np.repeat(gnd_mse, 1000), color='green')

    plt.title('Training Loss')
    plt.scatter(polyDegree, MSE,  color='black')
    plt.axis([-5, 30, -5, 225])

    MSE_test = np.zeros(1000)
    for i in range(0,1000):
      models[i] = np.poly1d(np.polyfit(x_test, y_test, i))
      pred_y = np.polyval(models[i], x_test)
      MSE_test[i] = ((y_test - pred_y) ** 2).mean(axis=None)
      polyDegree[i] = i

    gnd_y = np.polyval(reg, x_test)
    gnd_mse = ((y_test - gnd_y) ** 2).mean(axis=None)
    gnrd_x =np.linspace(0, 30, 1000)
    print(gnd_mse)
    plt.figure(3)
    plt.plot(gnd_x, np.repeat(gnd_mse, 30), color='green')
    print(polyDegree)
    print(MSE)

    plt.title('Validation Testing Loss')
    plt.scatter(polyDegree, MSE_test, color='black')
    plt.axis([-5,30,-5,225])
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2882: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-597c5bffc4d2> in <module>()
     50 MSE_test = np.zeros(1000)
     51 for i in range(0,1000):
---> 52   models[i] = np.poly1d(np.polyfit(x_test, y_test, i))
     53   pred_y = np.polyval(models[i], x_test)
     54   MSE_test[i] = ((y_test - pred_y) ** 2).mean(axis=None)

<__array_function__ internals> in polyfit(*args, **kwargs)

/usr/local/lib/python3.6/dist-packages/numpy/lib/polynomial.py in polyfit(x, y, deg, rcond, full, w, cov)
    599         raise TypeError("expected 1D vector for x")
    600     if x.size == 0:
--> 601         raise TypeError("expected non-empty vector for x")
    602     if y.ndim < 1 or y.ndim > 2:
    603         raise TypeError("expected 1D or 2D array for y")

TypeError: expected non-empty vector for x
```

SEARCH STACK OVERFLOW





Problem 2

This homework already took me more than 30 plus hours to do, just to get particually done with 1. With Coms 230 & Coms 311 they gave 20% for answers that were "I don't know" simply because they didn't want to waste the students time or the TA's time grading wrong answers and understood sometimes the student doesn't know. If you want to count that, that would overall help me out alot. I think for this class it should be somethng to consider since once again this home work was way toooo long and wasn't in anyway great for checking correct work. I had fun something to consider since once again this home work was way toooo long and wasn't in anyway great for checking correct work. I had fun with this homework but honestly the time it took to fully understand the concepts, hair pulling out, fustration, and other things it wasn't worth the time. I'll take the L on this section and Say:

I don't know.