9. Mobile App Store

Name: Lee Hong Yi (2223010)

Question 1a).

A graph with a red line and a red line

Description automatically generated

Equation of regression line obtained: user\_rating = (0.3917)price

Question 1b)

A white paper with writing on it

Description automatically generated

Value of n is 7194.

Question 1ci)

|  |  |
| --- | --- |
| Code | def model1\_grad\_descent(*learning\_rate* = 0.01, *stop\_diff* = 0.001, *iter* = 1000, *start\_val* = 0):      y = data["user\_rating"].values      x = data["price"].values      b = *start\_val* #starting value of b      rate = *learning\_rate* #size of step taken by grad descent model      epsilon = *stop\_diff* #stops model when difference between b is below this value      max\_iteration= *iter* #stops model when number of iterations taken exceeds value      diff = 100      iterations = 1      def func(*b*):          return 1/n \* np.sum((-*b* \* x + y)\*\*2)      def diff\_func(*b*):          return 1/n \* np.sum(-2 \* x \* (-*b* \* x + y))      while diff > epsilon and iterations < max\_iteration:          b\_new = b - rate \* diff\_func(b)          diff = abs(b\_new - b)  # new difference          if func(b\_new) == np.inf:              print("Failed to converge.")              break*;*          iterations += 1          b = b\_new  # new b value      print(f"== Results (Learning Rate: {*learning\_rate*}, Stop Value: {*stop\_diff*}, Starting Value: {*start\_val*}) ==")      print(f"Number of iterations is {iterations}\nThe local minimum occurs when b = {b}.\nMinimum error is {func(b)}\n")  model1\_grad\_descent(*learning\_rate* = 0.05) |
| Output | == Results (Learning Rate: 0.05, Stop Value: 0.001, Starting Value: 0) ==  Number of iterations is 14  The local minimum occurs when b = 0.39207641432075924.  Minimum error is 12.315001935889542 |

Question 1cii)

Equation of line obtained: user\_rating = (0.39207641432075924)price

Minimum error is obtained when the learning rate is 0.05.

Question 2a)

A graph with blue dots and a red line

Description automatically generated

Equation obtained: user\_rating = 3.479 + (0.02876)price

If price = 0.99, user\_rating = 3.5074724

Question 2b)

A white board with math equations

Description automatically generated

Question 2c)

|  |  |
| --- | --- |
| Code | def model2\_grad\_descent(*a\_next*=0, *b\_next*=0, *learning*=0.1, *stop\_val*=0.001, *max\_iter*=1000):      y = data["user\_rating"].values      x = data["price"].values      diff = 100      # iter = 0      init\_a = a\_next      init\_b = b\_next      function = lambda *a*, *b*: 1/n \* np.sum((-a - b\*x + y)\*\*2)      partial\_a = lambda *a*, *b*: 1/n \* np.sum(2\*a + 2\*b\*x - 2\*y)      partial\_b = lambda *a*, *b*: 1/n \* np.sum(-2\*x \* (-a - b\*x + y))      next\_func = function(a\_next, b\_next)      for iter in range(max\_iter):          current\_a = a\_next          current\_b = b\_next          current\_func = next\_func          a\_next = current\_a - learning \* partial\_a(current\_a, current\_b)          b\_next = current\_b - learning \* partial\_b(current\_a, current\_b)          next\_func = function(a\_next, b\_next)          change\_func = abs(next\_func - current\_func)          # print(f"Iteration {iter+1}, a = {a\_next}, b = {b\_next}, E(a,b) = {next\_func}, difference = {change\_func}")          if change\_func < stop\_val:              break      print(f"== Result (a = {init\_a}, b = {init\_b}, Learning Rate = {learning}, Epsilon = {stop\_val}) ==")  print(f"Number of iterations: {iter+1}\nLocal minima occurs at a = {a\_next}, b = {b\_next}\nE(a,b) is {next\_func}")  model2\_grad\_descent(3, 0, 0.05, 0.000000001) |
| Output | == Result (a = 3, b = 0, Learning Rate = 0.05, Epsilon = 1e-09) ==  Number of iterations: 102  Local minima occurs at a = 3.479152372034554, b = 0.028770272235997735  Minimum error is 2.2937798957259568 |

Question 2cii)

Equation obtained: user\_rating = 3.479152372034554 + (0.028770272235997735)price

Question 2d)

I start attempting to find the final values by assuming the initial values (a = 3, b = 2, Learning Rate = 0.01, Epsilon = 0.001). As this already achieved convergence, the focus shifted away from achieving convergence, and instead attempting to find the values which would yield the lowest error. To do this, I first attempted to tune the learning rate (the size of the step taken by the algorithm), decreasing it in a attempt to yield a lower error with the trade-off of increased compute time (increased iteration count). When attempting this, the minimum error increased from 2.3226 to 2.5358. Due to this, I deduced that in order to achieve the lowest possible error, I would have to increase the learning rate, white being careful to not take steps which are too big. I settled on a rate of 0.05, which returned a minimum error of 2.2984. This verified my deduction, and showed that a increase in learning rate would yield a lower error value. Then, I adjusted the starting point to be similar to the values obtained in Minitab earlier, hence, I start with the values (a =3, b = 0). While this caused a slight increase in the error value, I believed that it was negligible. After that, I tuned Epsilon (size of difference to stop below), decreasing it from 0.001 to 0.0001, and saw that it caused a decrease in the error (2.29885 to 2.29424). This showed me that decreasing the value of Epsilon would help us achieve a smaller error value, and hence I decided to continue on this path, decreasing it to 0.000000001. This further decreased the error value, from 2.294244 to 2.293779. I decided not to adjust any further, as further increasing the learning rate might result in the model no longer converging, and a further decrease in Epsilon might sharply increase the number of iterations needed to reach convergence, with a negligible effect on the minimum error (law of diminishing returns). Each step can be seen in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Attempt | Parameters | Results | Remarks |
| 1 | a = 3, b = 2, Learning Rate = 0.01, Epsilon = 0.001 | Number of iterations: 79  Local minima occurs at a = 3.2925123902455415, b = 0.049304183135299946  Minimum error is 2.3226596508629145 | Starting parameters. |
| 2 | a = 3, b = 2, Learning Rate = 0.001, Epsilon = 0.001 | Number of iterations: 151  Local minima occurs at a = 2.9437181391795515, b = 0.10181659149025603  Minimum error is 2.5358585373203844 | Increase in learning rate results in larger error.  Number of iterations doubled. |
| 3 | a = 3, b = 2, Learning Rate = 0.05, Epsilon = 0.001 | Number of iterations: 26  Local minima occurs at a = 3.4045422794257334, b = 0.03698203238692591  Minimum error is 2.2984007772942636 | Decrease in learning rate decreased error.  Number of iterations sharply decreased. |
| 4 | a = 3, b = 0, Learning Rate = 0.05, Epsilon = 0.001 | Number of iterations: 21  Local minima occurs at a = 3.400947279319027, b = 0.037376080044957696  Minimum error is 2.298856301592866 | Adjusted starting point, negligible effect on error. |
| 5 | a = 3, b = 0, Learning Rate = 0.05, Epsilon = 0.0001 | Number of iterations: 35  Local minima occurs at a = 3.455544410322144, b = 0.031367593754234625  Minimum error is 2.2942446096039846 | Tuned epsilon, lowering error value. |
| 6 | a = 3, b = 0, Learning Rate = 0.05, Epsilon = 1e-09 | Number of iterations: 102  Local minima occurs at a = 3.479152372034554, b = 0.028770272235997735  Minimum error is 2.2937798957259568 | Lowered epsilon again, smaller decrease in error value.  Large increase in iteration count. |

Question 3a)

I start the data collection by checking how the given data is like, using pandas’ describe function. Using the results of the function, I can see that there are some outliers in the data, especially in the price function, with 50% of the apps being free, and 75% being below 1.99. However, the maximum value present is $99.99, telling me that there are outliers present in this data. The same also applies for the rest of the variables present, with a few apps being outliers.

Next, I use a corrplot to see how correlated the variables are. I do this, so that I can see if there are any variables which are highly correlated, and thus should not be used for variable w. From the corrplot, I can see that there are no variables which are highly correlated, and hence, all of them can be used for variable w.

From here, I then decide which variable is best to take for w. Since the final value to be predicted is “user\_rating", I pick the variable which seems to have the most relation to it, being “rating\_count\_tot”. I picked this variable, as the other variables, “user\_rating\_ver”, and “rating\_count\_ver” are based on the most recent version of the app, they will not be able to give a clear picture of how the app has actually performed throughout it’s whole lifespan. In addition, I chose not to use the size\_bytes variable, as it depends on the features built into the app, and may not have much influence on the final rating.

From here, I decide to use scikitlearn’s MinMaxScaler to scale the data, putting all of the features used on the same scale. MinMaxScaler does this by subtracting the minimum value from each value in the dataset, so that the minimum becomes zero. The scaler then divides the data by the range of the dataset, to scale it between 0 and 1. Without it, it may result in inaccurate coefficient interpretation, and prevent the model from achieving convergence. Scaling the data will lead to improved convergence of the algorithm, and allow the model to perform better. I also drop the unused rows in the dataset to make it more convenient for me to work with.

I also remove apps which have extreme values on the “price” and “rating\_count\_tot” columns, as they cause the data to be extremely skewed, resulting in a poor fit for the rest of the apps. I remove apps which exceed the 95th percentile on both values, resulting in a final dataset which has 6618 rows. I take all of the rows in the dataset, as it will allow for all “normal” apps to be properly represented. Due to this, it is not feasible for me to paste all the rows into this word document. I have exported the scaled data to a excel file labelled “data\_scaled.xlsx”, which can be [opened here](https://d.docs.live.net/ccc66450af79773e/A-SCHOOL%20MATERIAL/polytechnic/2023_S1/AA_CA2%20Folder/MAI/data_scaled.xlsx), and is also present in my submission. I have also attached the first and last 5 rows of the dataset into this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **index** | **price** | **rating\_count\_tot** | **user\_rating** |
| **0** | 0.570815 | 0.443288 | 0.8 |
| **1** | 0.141631 | 0.171823 | 0.8 |
| **2** | 0.570815 | 0.164161 | 0.8 |
| **3** | 0.713877 | 0.131995 | 0.9 |
| **4** | 0.427754 | 0.654897 | 0.8 |
| … | … | … | … |
| **6613** | 0 | 0.0029563624 | 0.9 |
| **6614** | 0 | 0.0006245836 | 0.9 |
| **6615** | 0.284692 | 0.0003122918 | 0.9 |
| **6616** | 0 | 0.0017696536 | 0.9 |
| **6617** | 0 | 0.0000624584 | 1 |

Question 3b)

Since the equation for , we substitute it into the error formula. From there, we can use partial differentiation, and obtain the partial derivative for the formula, with respect to a, b, and c. This will allow us to find the rate of change of E(a, b, c) with respect to all variables.

A white paper with black text

Description automatically generated

Hence, the equations for the error function, and gradient descent algorithm are as follows:

Error function:

These equations are used in gradient descent to find the direction of steepest descent. By obtaining the values of the derivatives, we can then evaluate how much to change the value of the parameters of the model to minimize the error function. By taking the partial derivatives of the loss function with respect to each parameter, we can then determine how much the error will change if a small change is made to that parameter. In gradient descent, the information is used to iteratively update the parameters in a way that gradually reduces the error, leading to the parameters which can provide the best predictions.

Question 3c)

|  |  |
| --- | --- |
| Code | def model3\_grad\_descent(*a\_next*=3, *b\_next*=0, *c\_next* = 0, *learning*=0.01, *stop\_val*=0.0001, *max\_iter*=1000):      n = len(scaled\_data)      # print(n)      y = scaled\_data["user\_rating"].values      x = scaled\_data["price"].values      w = scaled\_data["rating\_count\_tot"].values      diff = 100      # iter = 0      init\_a = *a\_next*      init\_b = *b\_next*      init\_c = *c\_next*      function = lambda *a*, *b*, *c*: 1/n \* np.sum((-*a* - *b*\*x - *c*\*w + y)\*\*2)      partial\_a = lambda *a*, *b*, *c*: 1/n \* np.sum(2\**a* + 2\**b*\*x + 2\**c*\*w - 2\*y)      partial\_b = lambda *a*, *b*, *c*: 1/n \* np.sum(-2\*x \* (-*a* - *b*\*x - *c*\*w + y))      partial\_c = lambda *a*, *b*, *c*: 1/n \* np.sum(-2\*w \* (-*a* - *b*\*x - *c*\*w + y))      next\_func = function(*a\_next*, *b\_next*, *c\_next*)      for iter in range(*max\_iter*):          current\_a = *a\_next*          current\_b = *b\_next*          current\_c = *c\_next*          current\_func = next\_func  *a\_next* = current\_a - *learning* \* partial\_a(current\_a, current\_b, current\_c)  *b\_next* = current\_b - *learning* \* partial\_b(current\_a, current\_b, current\_c)  *c\_next* = current\_c - *learning* \* partial\_c(current\_a, current\_b, current\_c)          next\_func = function(*a\_next*, *b\_next*, *c\_next*)          change\_func = abs(next\_func - current\_func)          # print(f"Iteration {iter+1}, a = {a\_next}, b = {b\_next}, c = {c\_next},  E(a,b,c) = {next\_func}, difference = {change\_func}")          if change\_func < *stop\_val*:              break          elif np.isnan(next\_func) or next\_func == np.inf:              print("Failed to converge.")              break      print(f"== Result (a = {init\_a}, b = {init\_b}, c = {init\_c}, Learning Rate = {*learning*}, Epsilon = {*stop\_val*}) ==")      print(f"Number of iterations: {iter+1}\nLocal minima occurs at a = {*a\_next*}, b = {*b\_next*}, c = {*c\_next*}\nMinimum error is {next\_func} ")      return *a\_next*, *b\_next*, *c\_next*    model3\_a, model3\_b, model3\_c = model3\_grad\_descent(0,1,1,0.06,0.0000001) |
| Output | == Result (a = 0, b = 0, c = 0, Learning Rate = 0.06, Epsilon = 1e-07) ==  Number of iterations: 1022  Local minima occurs at a = 0.6320717195649185, b = 0.19389440405950017, c = 0.4599167180981849  Minimum error is 0.08789430028721607 |

Final Equation: user\_rating = 0.6320717195649185 + 0.19389440405950017(price) + 0.4599167180981849(rating\_count\_tot)

Code explanation:

The code expands on the original gradient descent algorithm code used in Question 2, making it suitable for MLR. It does this by adding the third equation (error function differenced with respect to c), and replacing the existing equations with the ones obtained in 3b. It first calculates the error value with the starting values, then runs a gradient descent loop to find the minimum error. The gradient descent loop stops when the minimum error is below the Epsilon set (stop\_val variable), then prints out the results.

Verifying Regression line:

We verify that the regression line obtained is correct by using a regression plane, as well comparing it to Minitab. From the regression plane (image attached below), we can see that the regression line obtained somewhat follows the trend of the data, with some data points being directly on the regression plane. We can also see that the regression plane somewhat follows the trend of the data, with a upward correlation, showing that the obtained results will be following the trend, but may not be equal to the actual result. In addition, the final equation seems to be relatively similar to the one obtained in Minitab, showing that the final equation is correct.

A screen shot of a graph

Description automatically generatedA black text on a white background

Description automatically generated