**Assignment 2**

**Instructions:**

* Type your answers in the spaces provided in this Word document. Your submission should not exceed 11 pages, including this page.
* Submit the *Declaration of Academic Integrity* before submitting your assignment.

--------------------------------------------------------------------------------------------------------------------

**Introduction**

Given a set of data points with at least one predictor and one continuous response variable, we want to construct a linear model to predict the response. This is the aim of **Linear Regression**, which is a supervised learning technique.

In the context of this assignment, measurements of 25 fishes of the species Smelt are collected. The data can be found in the file *fish.csv*. The following table lists the variables used in the file and their descriptions:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| *Weight* | Weight in 0.1gram |
| *Length* | Length of the fish in cm |
| *Width* | Width of the fish in cm |

The response variable is *Weight,* and the predictors are *Length* and *Width*.

**Simple Linear Regression (SLR)**

We will first build a SLR model using *Length* as the predictor to predict *Weight*.

In SLR notations, let:

= predictor value of the *i*-th data point

= actual response value of the *i*-th data point

= predicted response value of the *i*-th data point based on model

Thus, , where values of *a* (intercept) and *b* (slope) are to be determined.

The squared error of the *i*th prediction is . Errors (also known as residuals) are squared to remove the signs, so that errors of opposite signs do not cancel out each other, giving the false impression of small aggregated errors.

Then, we define **Error function** as the mean of squared-error (of the whole data set):

We want to find the values of *a* and *b* such that the Error function is **minimised**.

The resultant equation will give the best-fit line that passes through the data points.

**MODEL 1: SLR with intercept *a* fixed ⇒**  (25 marks)

We will first build a SLR model to predict *Weight* (*y*) using *Length* (*x*) as the predictor.

Suppose it is believed that weight is directly proportional to length. This means that  is a constant multiple of  and . Then, in the SLR model, we will only need to determine slope *b*.

(a) Express Error function in terms of *b* only. Hence, derive .

|  |
| --- |
| Express Error Function E(b) in terms of b only.  Derive |

(b) Use univariate gradient descent algorithm to find the value of *b* for which is at its minimum. Write your Python code in a single cell and copy-paste your code below.

|  |
| --- |
| # Define x and y  x = fish['Length'].values  y = fish['Weight'].values  model1\_b = 1 # Starting value of b  rate = 0.0001 # Set learning rate  epsilon = 0.00001 # Stop algorithm when absolute difference between 2 consecutive x-values is less than epsilon  diff = 1 # difference between 2 consecutive iterates  max\_iter = 1000 # set maximum number of iterations  iter = 1 # iterations counter  deriv = lambda b: 1/25 \* np.sum(-2 \* x \* ( -model1\_b \* x + y)) ## derivative of loss function  error = lambda b: 1/25 \* np.sum((y - model1\_b \* x)\*\*2) # loss function  # Gradient Descent Algorithm of E(b), finding the minima of the loss function  while diff > epsilon and iter < max\_iter:      b\_new = model1\_b - rate \* deriv(model1\_b)      print("Iteration ", iter, "\tb-value is: ", b\_new,"\tE(b) is: ", error(b\_new), "\tE'(b) is:", deriv(b\_new))      diff = abs(b\_new - model1\_b) ## new difference      iter = iter + 1 ## new iteration      model1\_b = b\_new ## new b value    print("The local minimum occurs at: ", model1\_b, '\nE(b) is: ', error(model1\_b)) |

(c) Describe the changes and decisions you made on the parameters for your solution to reach convergence.

|  |
| --- |
| From my initial baseline model, the error function produced a very high value, 851.3107504072279, while having little iterations, 27, for Model 1 SLR to reach convergence. This can be interpreted that the model produces very large residuals which can be determined that the model is a bad fit to the training data.  Since the error functions produce a high value while there are little iterations to reach convergence, my attempt to change the parameters to reduce error function :   1. Set epsilon/criterion to 0.00001    * A smaller threshold between two consecutive weights, to have better expected accuracy.   epsilon = 0.00001 # stopping criterion   1. Set learning rate to 0.0001    * For b/weight to take smaller steps per iteration, perform better optimization.   rate = 0.0001 # Set learning rate |

(d) Describe your MODEL 1 by filling the information below.

|  |
| --- |
| Final MODEL 1 equation is:  Minimum value of Error function is: 851.3093998441157  Number of iterations ran to reach convergence: 379    Figure 1 – Bivariate plot with Model 1 SLR  While the error function **did decrease**, it decreased in a **very small margin** compared to my initial baseline. In Figure 1, it can be observed that there are very large residuals produced on the regression line, where , which indicates that the data does not fit well to the gradient descent process. This perhaps indicates that *Weight* (*y*) might not have a directly proportional relationship with *Length* (*x*) which can be concluded that Model 1 SLR *Weight* (*y*). |

**MODEL 2: SLR ⇒**  (25 marks)

Now we apply the SLR model where both intercept *a* and slope *b* are to be determined, when predicting *Weight* (*y*) using *Length* (*x*) as the predictor.

(a) Express Error function in terms of *a* and *b*. Hence, derive and .

|  |
| --- |
| Express Error function in terms of *a* and *b*, where    Derive  Derive |

(b) Use gradient descent algorithm to find the values of *a* and *b* for which is at its minimum. Write your Python code in a single cell and copy-paste your code below.

|  |
| --- |
| # Define x and y  x = fish['Length'].values  y = fish['Weight'].values  next\_b = (y[0] - y[-1]) / (x[0] - x[-1]) # Initial point of b  next\_a = (y[0] - next\_b \* x[0]) # Initial point of a  alpha = 0.005 # Learning rate  epsilon = 0.0001 # Stopping criterion constant  max\_iters = 25000 # Maximum number of iterations  # Partial derivatives and function  partialf\_a = lambda a,b: 1/25 \* np.sum(2\*a + 2 \* b \* x - 2 \* y)  partialf\_b = lambda a,b: 1/25 \* np.sum(-2 \* x \* (-a -b\*x+y))  func = lambda a,b: 1/25 \* np.sum((y - (a + b\*x))\*\*2)  next\_func = func(next\_a,next\_b) # Initial value of function  for n in range(max\_iters):      current\_a = next\_a      current\_b = next\_b      current\_func = next\_func      next\_a = current\_a-alpha\*partialf\_a(current\_a,current\_b) # update of a      next\_b = current\_b-alpha\*partialf\_b(current\_a,current\_b) # update of b      next\_func = func(next\_a,next\_b)      change\_func = abs(next\_func-current\_func) # stopping criterion: values of function converge      print("Iteration",n+1," a =",next\_a,", b =",next\_b,",\tE(a,b) =",next\_func)      if change\_func < epsilon:          break |

(c) Describe the changes and decisions you made on the parameters for your solution to reach convergence.

|  |
| --- |
| From my initial baseline model, the iteration count reached the limit, which can be concluded that the model did not reached convergence. Model 2 SLR produced an error function of 404.16225317287956, which is better than the results produced in Model 1 SLR.  In order to reduce the error function , I will attempt to change the parameters:   1. Set the initial values for and using the gradient and intercept of the two furthest points, using the basic concept of linear equations .    * I noticed that and deviate a lot from the initial start point, which indicates that the initial starting point matters in order to reach convergence faster.    * Calculate initial point of b using, .    * Calculate initial point of a using, .   next\_b = (y[0] - y[-1]) / (x[0] - x[-1]) # Initial point of b  next\_a = (y[0] - next\_b \* x[0]) # Initial point of a   1. Increase learning rate to 0.0005    * Reach convergence faster since model initially hit max iteration limit.   alpha = 0.005 # Learning rate   1. Decrease criterion/epsilon to 0.0001    * A drawback to reducing learning rate is a trade off in accuracy, as such I would like to decrease the criterion value in order to mitigate any flaw in accuracy.   epsilon = 0.0001 # Stopping criterion constant |

(d) Describe your MODEL 2 by filling the information below.

|  |
| --- |
| Final MODEL 2 equation is:  Minimum value of Error function is: 137.98078414287622  Number of iterations ran to reach convergence: 21533    Figure 2 – Bivariate plot with Model 2 SLR  The error function decreased by a significant margin, while number of iterations also decreased down to a point where convergence can occur. A low error functions indicates that there are small residuals when fitting the model to the training data. On figure 2, it can be observed that the data points are about the regression line, which indicates a good fitting of the data to the gradient descent process. Overall, it can be concluded that Model 2 SLR has a good fit to the data. |

**Conclusion on SLR** (15 marks)

(a) Using Python (or other software), in a single figure, plot the data points (scatterplot) together with the linear lines representing the two models. Insert the figure below and describe what you observe regarding the location of the data and the linear lines.

|  |
| --- |
| Figure 3 - Bivariate Plot with SLR  Model 1 SLR, the regression line does not fit well to the training data, which can be seen that the line does not follow the direction of the data points. This is further supported by the very large residuals from the regression line, which explains the very high value in the error function .  Whereas Model 2 SLR adding a parameter , the regression line fits well to the training data, as seen that the line follows the direction of the data point. It can be seen that the data points are located around Model 2 regression line, which explains the low value in the error function . |

(b) In a linear regression model, the constant  is commonly interpreted as the value of the response variable when the predictor variable is zero. In your Model 2, can you interpret your value of  as such? Explain.

|  |
| --- |
| Assuming predictor variable ，  , which seems impossible for a fish to have negative weight. As such, we reject the claim that constant in Model 2 is commonly interpreted as the value of the response variable when the predictor variable is zero, since has to be a positive value.  A better interpretation would be a graph **similar** to a neural network activation function RELU (Rectified Linear Unit), such that the only values we can accept is when , such that will always be positive. |

**MODEL 3: MLR ⇒**  (25 marks)

We can extend the SLR model to include more predictors. A linear regression model with more than 1 predictor is called **Multiple Linear Regression** (MLR) model.

Apply the MLR model where intercept *a*, and slopes *b* and *c* are to be determined, when predicting *Weight* (*y*) using *Length* (*x*) and *Width* (*w*) as the predictors.

(a) Explain how gradient descent algorithm can be extended for MODEL 3.

|  |
| --- |
| Multivariate Linear Regression is defined as follow:  Error Function, expressed in terms of , is defined as follows:  Step 0: Set a learning rate > 0 and an initial point , and compute .  Step 1: At n-th point , compute , and  .  Step 2: Update to the -th point using ,  and .  Step 3: Repeat steps 1 and 2 until a stopping criterion is reached.  Stopping criterion here refers to:   * Convergence occurs - the difference between two consecutive error function smaller than a fixed constant * the number of iterations reached the max iteration limit   Derivative with respect to ,  Derivative with respect to ,  Derivative with respect to , |

(b) Use gradient descent algorithm to find the values of *a*, *b* and *c* for which Error function is at its minimum. Write your Python code in a single cell and copy-paste your code below.

|  |
| --- |
| # Define w, x and y  w = fish['Width'].values  x = fish['Length'].values  y = fish['Weight'].values  next\_c = (y[0] - y[-1]) / (w[0] - w[-1]) # Initial point of c  next\_b = (y[0] - y[-1]) / (x[0] - x[-1]) # Initial point of b  next\_a = (y[0] - next\_b\*x[0] - next\_c\*w[0]) # Initial point of a  alpha = 0.005 # Learning rate  epsilon = 0.0001 # Stopping criterion constant  max\_iters = 50000 # Maximum number of iterations  # Partial derivatives and function  partialf\_a = lambda a,b,c: 1/25 \* np.sum(2\*a + 2 \* b \* x + 2\*c\*w - 2 \* y)  partialf\_b = lambda a,b,c: 1/25 \* np.sum(-2 \* x \* (-a -b\*x -c\*w+y))  partialf\_c = lambda a,b,c: 1/25 \* np.sum(-2 \* w \* (-a -b\*x -c\*w+y))  func = lambda a,b,c: 1/25 \* np.sum((y - (a + b\*x + c\*w))\*\*2)  next\_func = func(next\_a,next\_b,next\_c) # Initial value of function  for n in range(max\_iters):      current\_a = next\_a      current\_b = next\_b      current\_c = next\_c      current\_func = next\_func      next\_a = current\_a-alpha\*partialf\_a(current\_a,current\_b,current\_c) # update of a      next\_b = current\_b-alpha\*partialf\_b(current\_a,current\_b,current\_c) # update of b      next\_c = current\_c-alpha\*partialf\_c(current\_a,current\_b,current\_c) # update of c      next\_func = func(next\_a,next\_b,next\_c)      change\_func = abs(next\_func-current\_func) # stopping criterion: values of function converge      print("Iteration",n+1," a =",next\_a,", b =",next\_b,       ', c =', next\_c, ", E(a,b,c) =",next\_func)      if change\_func < epsilon:          break |

(c) Describe the changes and decisions you made on the parameters for your solution to reach convergence.

|  |
| --- |
| From my initial baseline model, the iteration count reached the limit, which can be concluded that the model did not reached convergence. Model 3 MLR produced a MSE (Mean Squared Error) of 122.36627441191841.  To reduce error function , I will attempt to change the parameter:   1. Set the initial and using the gradient and intercept of the two furthest points, using the basic concept of linear equations .    * I noticed that and deviate a lot from the initial start point, which indicates that initial starting point matters to reach convergence faster.    * Calculate initial point of b using, .    * Calculate initial point of c using, .    * Calculate initial point of a using, .   next\_c = (y[0] - y[-1]) / (w[0] - w[-1]) # Initial point of c  next\_b = (y[0] - y[-1]) / (x[0] - x[-1]) # Initial point of b  next\_a = (y[0] - next\_b\*x[0] - next\_c\*w[0]) # Initial point of a   1. Increase learning rate to 0.005    * Reach convergence faster since model initially hit max iteration limit.   alpha = 0.005 # Learning rate   1. Decrease criterion/epsilon to 0.0001    * A drawback to reducing learning rate is a trade off in accuracy, as such I would like to decrease the criterion value in order to mitigate any flaw in accuracy.   epsilon = 0.0001 # Stopping criterion constant |

(d) Describe your MODEL 3 by filling the information below.

|  |
| --- |
| Final MODEL 3 equation is:  Minimum value of Error function is: 47.290389726153975  Number of iterations ran to reach convergence: 36531    Figure 4 - Model 3 MLR  The error function decreased by a significant margin, while the number of iterations also decreased down to a point where convergence can occur. A low error function indicates that there are small residuals when fitting the model to the training data. As seen in Figure 4, Model 3 MLR is more of a plane rather than a line, visualising the works of multivariate linear regression. Furthermore, when mapping a line chat to , , and it shows that equation is fitting to the data points nicely, having no extreme residuals to be found. Overall, it can be concluded that has a good fit to the data. |

**Conclusion** (10 marks)

(a) David used gradient descent algorithm to find the 3 models. Next, he computed the predicted weights using the 3 models for all the data points in the dataset. He noticed that for one of the data points, the error of the predicted weight in Model 1 from the actual weight is the smallest, compared to the other 2 models. Is this possible, assuming he has done his gradient descent algorithm correctly? Explain.

|  |
| --- |
| It is possible, as that one of the data points could produce very little residual for Model, which could mean that one fish has its length directly proportional to weight, . However, while it is only a small error for Model 1 for that one single data point, generally Model 1 produced the largest error when performing gradient descent on the entire dataset. |

(b) Compare the 3 models. Which model will you use to predict weight in this context? Explain.

|  |
| --- |
| In my opinion, model 3 MLR is the most suitable model to predict weight in this context. Model 3 MLR produced the smallest error function of predicted weight, , among all of the models, which makes the model more reliable in making predictions due to predicting lesser residuals, rather than a Model 1 SLR with a high error function which the produces the smallest error of the predicted weight for that **one** fish.    Figure 5 - Prediction Error Plot  Observed from Figure 5, which shows Model MLR 3 has the least number of residuals among all of the models, where there is little deviation between the predicted point and the actual values, as seen by the distance between the points from the scatterplot and the line. Whereas for Model 1, the points seem to not follow the direction of the line, which indicates the model unreliable in making predictions. Furthermore, the results produced from Model 3 has the highest among all of the models, which explains that majority of the predicted has little variation from the actual . In conclusion, David should use Model 3 MLR as it is the most reliable model to predict fishes’ weight in this context, where reliable refers to producing a low error function. |