**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 .

|  |
| --- |
|  |

(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.

|  |
| --- |
| def gradient\_descent\_b(      df: pd.DataFrame, b\_start: float,      alpha: float = 0.01, epsilon: float = 0.0001, max\_iter: int = 100  ):      def E(df: pd.DataFrame, b: float):          y\_true = df['Weight']          y\_hat = b \* df['Length']          err = y\_true - y\_hat          return (err \*\* 2).mean()      def E\_b(df:pd.DataFrame, b: float):          y\_true = df['Weight']          y\_hat = b \* df['Length']          err = y\_true - y\_hat          return -2 \* (err \* df['Length']).mean()      def gradient\_descent(          df: pd.DataFrame,          f: Callable, f\_b: Callable, b\_start: float,          alpha: float = 0.01, epsilon: float = 0.0001,          max\_iter: int = 100      ):          b = b\_start          difference = epsilon + 1          e = f(df, b)          print(f'i=0; b={b:10.2f}; e={e:10.2f}')          for i in range(1, max\_iter + 1):              b\_next = b - alpha \* f\_b(df, b)              e\_next = f(df, b\_next)              difference = abs(e\_next - e)              print(f'\ri={i}; b={b:10.5f}; e={e:10.5f}', end='')              if difference < epsilon:                  break              b = b\_next              e = e\_next      gradient\_descent(          df=df, b\_start=b\_start,          f=E, f\_b=E\_b,          alpha=alpha, epsilon=epsilon, max\_iter=max\_iter)  gradient\_descent\_b(df=df, b\_start=8, alpha=0.004, epsilon=0.00001, max\_iter=100) |

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

|  |
| --- |
| Different values of b\_start were tested: 0, 1, 2, 8, 10  Different values of alpha were tested: 0.001, 0.004, 0.005, 0.007, 0.01  Different values of epsilon were tested: 0.001, 0.0001, 0.00001 |

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

|  |
| --- |
| Final MODEL 1 equation is:  Minimum value of Error function is: 851.30938  Number of iterations ran to reach convergence: 5 |

**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 .

|  |
| --- |
|  |

(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.

|  |
| --- |
| def gradient\_descent\_a\_b(      df: pd.DataFrame,      a\_start: float, b\_start: float,      alpha: float = 0.01, epsilon: float = 0.0001, max\_iter: int = 100  ):      def E(df: pd.DataFrame, a: float, b: float):          y\_true = df['Weight']          y\_hat = b \* df['Length'] + a          err = y\_true - y\_hat          return (err \*\* 2).mean()        def E\_a(df: pd.DataFrame, a: float, b: float):          y\_true = df['Weight']          y\_hat = b \* df['Length'] + a          err = y\_true - y\_hat          return -2 \* err.mean()      def E\_b(df:pd.DataFrame, a: float, b: float):          y\_true = df['Weight']          y\_hat = b \* df['Length'] + a          err = y\_true - y\_hat          return -2 \* (err \* df['Length']).mean()      def gradient\_descent(          df: pd.DataFrame,          f: Callable, f\_a: Callable, f\_b: Callable,          a\_start: float, b\_start: float,          alpha: float = 0.001, epsilon: float = 0.0001,          max\_iter: int = 100      ):          a = a\_start          b = b\_start          e = f(df=df, a=a, b=b)          difference = epsilon + 1          print(f'i=0; a={a:10.2f}; b={b:10.2f}; e={e:10.2f}')          for i in range(1, max\_iter + 1):              pd\_a = f\_a(df=df, a=a, b=b)              pd\_b = f\_b(df=df, a=a, b=b)              a\_next = a - alpha \* pd\_a              b\_next = b - alpha \* pd\_b              e\_next = f(df=df, a=a\_next, b=b\_next)              difference = abs(e\_next - e)              a = a\_next              b = b\_next              e = e\_next              print(f'\ri={i}; a={a:10.5f}; b={b:10.5f}; e={e:10.5f}', end='')              if min(difference, pd\_a \*\* 2 + pd\_b \*\* 2) < epsilon:                  break      gradient\_descent(          df=df,          f=E, f\_a=E\_a, f\_b=E\_b,          a\_start=a\_start, b\_start=b\_start,          alpha=alpha, epsilon=epsilon, max\_iter=max\_iter      )  gradient\_descent\_a\_b(      df=df, a\_start=-275, b\_start=35,      alpha=0.004, epsilon=0.001, max\_iter=100  ) |

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

|  |
| --- |
| Different values for alpha were tested: 0.001, 0.004, 0.005, 0.007, 0.01  Different values for a\_start were tested: 0, 20, -100, -200, -250, -275  Different values for b\_start were tested: 0, 10, 30, 35  Different values for epsilon were tested: 0.0001, 0.001 |

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

|  |
| --- |
| Final MODEL 2 equation is:  Minimum value of Error function is: 138.22782  Number of iterations ran to reach convergence: 4 |

**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.

|  |
| --- |
| Chart, scatter chart  Description automatically generated  The Model 2’s prediction line fits the data points better than Model 1’s. I.e., the data points are closer to the orange line than the green one.  Model 1 is too simple a model for this regression problem as it assumes no y-intercept (a=0). Therefore, Model 1 underfits the data and produces a greater error than Model 2. |

(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.

|  |
| --- |
| No. For model 2, . That would imply that a fish of length 0cm would weigh approximately -27.5g (unit is 0.1g). It is erroneous to interpret this way as weight cannot be negative. Furthermore, a fish of 0cm length should logically have 0.0g weight as well (No length => No weight).  This contradiction arises as a result of extrapolation. The predictions of the linear regression models are only valid for interpolation (predicting within the bounds of the known values). |

**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.

|  |
| --- |
| The error function now includes a third term (). Hence, , , and have to be rederived.  For each iteration, , and are updated using their corresponding update rules simultaneously (. The stopping criterion will also be modified to consider as part of the squared vector norm stopping criterion. |

(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.

|  |
| --- |
| def gradient\_descent\_a\_b\_c(      df: pd.DataFrame,      a\_start: float, b\_start: float, c\_start: float,      alpha: float = 0.01, epsilon: float = 0.0001, max\_iter: int = 100  ):      def E(df: pd.DataFrame, a: float, b: float, c: float):          y\_true = df['Weight']          y\_hat = a + b \* df['Length'] + c \* df['Width']          err = y\_true - y\_hat          return (err \*\* 2).mean()        def E\_a(df: pd.DataFrame, a: float, b: float, c: float):          y\_true = df['Weight']          y\_hat = a + b \* df['Length'] + c \* df['Width']          err = y\_true - y\_hat          return -2 \* err.mean()        def E\_b(df:pd.DataFrame, a: float, b: float, c: float):          y\_true = df['Weight']          y\_hat = a + b \* df['Length'] + c \* df['Width']          err = y\_true - y\_hat          return -2 \* (err \* df['Length']).mean()        def E\_c(df:pd.DataFrame, a: float, b: float, c: float):          y\_true = df['Weight']          y\_hat = a + b \* df['Length'] + c \* df['Width']          err = y\_true - y\_hat          return -2 \* (err \* df['Width']).mean()        def gradient\_descent(          df: pd.DataFrame,          f: Callable, f\_a: Callable, f\_b: Callable, f\_c: Callable,          a\_start: float, b\_start: float, c\_start: float,          alpha: float = 0.001, epsilon: float = 0.0001,          max\_iter: int = 100      ):          a = a\_start          b = b\_start          c = c\_start          difference = epsilon + 1          e = f(df, a, b, c)          print(f'i=0; a={a:10.2f}; b={b:10.2f}; c={c:10.2f}; e={e:10.2f}')          for i in range(1, max\_iter + 1):              pd\_a = f\_a(df, a=a, b=b, c=c)              pd\_b = f\_b(df, a=a, b=b, c=c)              pd\_c = f\_c(df, a=a, b=b, c=c)              a\_next = a - alpha \* pd\_a              b\_next = b - alpha \* pd\_b              c\_next = c - alpha \* pd\_c              e\_next = f(df, a\_next, b\_next, c\_next)              difference = abs(e\_next - e)                a = a\_next              b = b\_next              c = c\_next              e = e\_next              print(f'\ri={i}; a={a:10.5f}; b={b:10.5f}; c={c:10.5f}; e={e:10.5f}', end='')              if min(difference, pd\_a \*\* 2 + pd\_b \*\* 2 + pd\_c \*\* 2) < epsilon:                  break        gradient\_descent(df=df,          f=E, f\_a=E\_a, f\_b=E\_b, f\_c=E\_c,          a\_start=a\_start, b\_start=b\_start, c\_start=c\_start,          alpha=alpha, epsilon=epsilon, max\_iter=max\_iter)  gradient\_descent\_a\_b\_c(      df=df,      a\_start=-210, b\_start=20, c\_start=60,      alpha=0.004, epsilon=0.001, max\_iter=100  ) |

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

|  |
| --- |
| Different values for alpha were tested: 0.001, 0.004, 0.005, 0.007, 0.01  Different values for epsilon were tested: 0.0001, 0.001  Different values for a\_start were tested: 0, -200, -210, -250  Different values for b\_start were tested: 0, 10, 20  Different values for c\_start were tested: 0, 20, 40, 60, 80 |

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

|  |
| --- |
| Final MODEL 3 equation is:  Minimum value of Error function is: 46.54312  Number of iterations ran to reach convergence: 4 |

**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.

|  |
| --- |
| Yes, this is possible. Models 2 and 3 fit the data better than Model 1, and therefore should yield lower errors than Model 1 most of the time.  However, there may be outliers in the data which are closer to the prediction line of Model 1 than the other 2 Models. While this is not impossible, it is rare and infrequently encountered. |

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

|  |
| --- |
| I would use Model 3.  Logical Argument: Model 3 takes both length and width into account when predicting weight. Models 1 and 2 only take length into account. Fish may be short or long, narrow or wide. Both variables affect the fish’s weight and are not necessarily interdependent, hence Model 3 is better in this context.  Numerical Argument: Model 3 also has the lowest mean squared error among the 3 models. |