**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, the transaction price of 30 flats in the same district of Singapore are collected. The data can be found in the file *housing\_price.csv*.

The response variable is *price per square metre* (measured in $ in thousands)*,* and the predictors are *inverse age of flat* (measured in year-1)and *inverse* *distance to the nearest MRT station* (measured in km-1). The *inverse age of flat* and *inverse* *distance to the nearest MRT station* are derived fields.

**Simple Linear Regression (SLR)**

We will first build a SLR model using *inverse* *distance to the nearest MRT station* as the predictor to predict *price per square metre*.

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 *price per square metre* (*y*) using *inverse distance to the nearest MRT station* (*x*) as the predictor.

Suppose it is believed that price is proportional to inverse distance. 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 .

|  |
| --- |
| To find E’(b), we need to perform partial differentiation using chain rule.    where m is the number of data instances, is the actual response value (price per square meter) for the ith data point, b is the slope, is the predictor value for the ith  data point |

(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.

|  |
| --- |
| b = 0.8132  import pandas as pd  df = pd.read\_csv('housing\_price.csv')  def costFunction(df, predictor, target , grad):      squarederrors = (df[target] - grad \* df[predictor])\*\*2      return sum(squarederrors) / len(df)    def partialDerivative(df, predictor, target, grad):      """      Returns the partial derivative of cost function      """      return -2/len(df) \* sum(df[predictor]\*(df[target] - grad \* df[predictor]))  x = 2 # Starting value of x  rate = 0.007 # Set learning rate  epsilon = 0.001 # Stop algorithm when absolute difference between 2 consecutive x-values is less than epsilon  diff = 1 # difference between 2 consecutive iterates  max\_iter = 15 # set maximum number of iterations  iter = 1 # iterations counter  while diff > epsilon and iter < max\_iter:      x\_new = x - rate \* partialDerivative(df, 'inverse distance to the nearest MRT station (km-1)' , 'price per square metre ($ in thousands)'  , x)      print("Iteration ", iter, ": b-value is: ", round(x\_new,2),"E(b) is: ", costFunction(df,  'inverse distance to the nearest MRT station (km-1)' , 'price per square metre ($ in thousands)' , x\_new ) )      diff = abs(x\_new - x)      iter = iter + 1      x = x\_new    print("The local minimum occurs at: ", round(x,7))  (df is the data ingested by pandas.read\_csv() ) |

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

|  |
| --- |
| **EPSILON**  I have attempted to optimize the epsilon parameter, which stops the algorithm when absolute difference between 2 consecutive x values is less then epsilon.   * If we set the epsilon parameter too small (example: 0.0000001), this means that the algorithm continues to run longer (More iterations), this is because a smaller difference between 2 consecutive b values is required for convergence. * If we set epsilon too high, gradient descent may not even converge as value of b may be selected inaccurately and far away from the minimum. This is because too large of a difference might cause algorithm to halt, where an inaccurate/nonoptimal b value might be chosen. B Values can also be quite different among iterations. * We need to select an epsilon value that is not too low or too high.   According to Andrew Ng a famous American Computer Scientist [1], ideally, convergence occurs if E(b) decreases by less than 0.001 in one iteration. Hence epsilon parameter is set to 0.001 to allow gradient descent to converge optimally. As I wanted the model to have high accuracy, the epsilon value I adjusted to is not too small or too large, avoiding the problems mentioned above.  [1] <https://www.coursera.org/learn/machine-learning>  **LEARNING RATE**  I have attempted to change Learning Rate parameter (alpha) for my solution to reach convergence. I will weigh the benefits and shortcomings of the different learning rates to pick.   * For example, if we set learning rate to **0.02**, we can see from figure below that from the first few iterations, E(b) does not decrease on every iteration but increases. Furthermore, b value changes drastically. These suggest that the model diverges away from the optimal solution, and **suggesting that learning rate is too high**, that gradient descent does not reach convergence, but misses/overshoots the minimum. To cite values, we can see that cost function increases from 112 to 1724 in a matter of 14 iterations. * Reducing the learning rate would enable the model to take smaller learning steps to the optimal solution of the cost function.      * If the learning rate is too small, it can be very slow to converge. For example, if we set learning rate to **0.001**, if we look at iteration 20 and onwards (figure below), the decrease/improvement of cost function is not very significant from iteration to iteration. This suggests that the **learning rate is too small**. Furthermore, it takes 45 **iterations to finally converge, which can be slow and not ideal.**     **OPTIMAL LEARNING RATE**   * To find the optimal learning rate, we observed that from the previous 2 evaluation shows that the optimal learning rate is between 0.001 and 0.02.   **Adjusting α = 0.007 through trial and error, we observe the following:**   * Convergence occurs at 7 iterations * The cost function decreases every iteration unlike the case for α=0.02. * The decrease in cost function for every iteration is more significant unlike the case for α=0.001. It also does not take so many iterations to converge. * We can conclude that gradient descent algorithm has converged to the optimal solution of the cost function.     **MAX ITERATIONS**  I have also properly chosen the maximum number of iterations. We can plot the error cost against number of iterations to properly pick number of iterations. From this plot, we can see that initially the error reduces significantly, but as iterations increase beyond 15 iterations, there seems to be no significant improvement and not much error reduction after that. And the curve seems to ‘flatten’ out after 15 iterations, suggesting that the cost function is decreasing negligibly. To decide the optimal number of iterations, we want to find the point where value of Mean squared error starts to decrease negligibly, which is at 15.  Hence, we can conclude that a good number of max\_iter = 15. However, the model converged at 7 iterations    **FINAL PARAMETERS**   1. MAX ITERATIONS: 15 2. Learning rate: 0.007 3. EPSILON: 0.001 4. Starting Gradient: 2 |

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

|  |
| --- |
| Final MODEL 1 equation is: = 0.8132  Minimum value of Error function is: 19.4671046132125 ≈ 19.5 (3sf)  Number of iterations ran to reach convergence: 7 |

**MODEL 2: SLR ⇒**  (25 marks)

Now we apply the SLR model where both intercept *a* and slope *b* are to be determined, when predicting *price per square metre* (*y*) using *inverse* *distance to the nearest MRT station* (*x*) as the predictor.

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

|  |
| --- |
| Through partial derivatives and chain rule:    where m is the number of data instances, is the actual response value (price per square meter) for the ith data point, b is the slope, is the predictor value (inverse distance) for the ith  data point, and a is the intercept |

(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.

|  |
| --- |
| a = 5.6319, b= 0.3091  import pandas as pd  df = pd.read\_csv('housing\_price.csv')  predictor = 'inverse distance to the nearest MRT station (km-1)'  targ = 'price per square metre ($ in thousands)'  def costFunction(df, grad, a,predictor = predictor, target=targ ):      squarederrors = (df[target] - grad \* df[predictor] -a )\*\*2      return sum(squarederrors) / len(df)  def partial\_diff\_a(df,b  ,a, predictor = predictor, target = targ):      return (-2 \* sum(df[target] - b\*df[predictor]  - a))/len(df)  def partial\_diff\_b(df,b  ,a, predictor = predictor, target =targ):      return (-2 \* sum(df[predictor]\*(df[target] - b\*df[predictor]  - a)))/len(df)  a = 5  grad = 2  rate = 0.015  # Set learning rate  epsilon = 0.0001  diff = 1  # difference between 2 consecutive iterates  max\_iter = 120  iter = 1 # iterations counter  cost = costFunction(df, grad,a)  while  iter < max\_iter and diff > epsilon:      b\_new = grad - rate \* partial\_diff\_b(df  , grad , a)      a\_new = a - rate \* partial\_diff\_a(df  , grad , a )      cost\_new =  costFunction(df, b\_new , a\_new )      print("Iteration ", iter  , 'a-value is :' , a\_new, ';',": b-value is: ", b\_new,"E(a,b) is: ", cost\_new)      diff = abs(cost\_new -  cost)      iter = iter + 1      cost = cost\_new      grad = b\_new      a = a\_new  print("The local minimum for a:" , a , 'b:', grad) |

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

|  |
| --- |
| **LEARNING RATE:**  I have attempted to change Learning Rate parameter (alpha) for my solution to reach convergence. I will weigh the benefits and shortcomings of the different learning rates to pick.   * If we set learning rate **to 0.02**, we can see from figure below that from the first few iterations, E(a,b) does not decrease on every iteration but increases rapidly. This leads to divergent outcomes, diverging away from optimal solution, suggesting that learning rate is too high, and gradient descent does not reach convergence, overshoots the minimum. For example, the cost function increases from 183 to 6379 in a matter of 15 iterations. * Ideally, we want the cost function to decrease every iteration.      * Reducing the learning rate would enable the model to take smaller learning steps to the optimal solution of the cost function. * Next, if the learning rate is like **0.01**, we can see that the decrease in cost function is not very significant, and the number of iterations needed for convergence is 204, which may be quite slow. We need to pick a higher learning rate as 0.01 results in slow convergence.     **OPTIMAL LEARNING RATE:**   * To find the optimal learning rate, our previous evaluation suggested that the optimal learning rate is around 0.01 and 0.02. * After adjusting through trial and error, the optimal learning rate is **0.015**, and we note that convergence occurs at around 148 iterations. * Optimal learning rate as it only requires 148 iterations to converge unlike alpha = 0.01 which takes 204 iterations. * Cost function decreases on every iteration unlike alpha = 0.02.     **MAX ITERATIONS**   * Optimal number of iterations seems to be around 120, as after that decrease in cost function seems to be less significant. Hence, max iter is chosen at 120     We can conclude that gradient descent algorithm has converged to optimal solution of the cost function.  **STARTING VALUE OF A & B**  A is selected to start at 5, and B is to start at 2. Cost function for linear regression is convex, hence it only has one minimum point, hence random initialization would generally lead to the same outcome of minimum point.  **EPSILON**   * Chosen to be 0.0001 to get the optimal value for a and b, at the same time minimizing the cost function |

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

|  |
| --- |
| Final MODEL 2 equation is: =5.6319 + 0.3091  Minimum value of Error function is: 0.4696 ≈0.470 (3sf)  Number of iterations ran to reach convergence:120 |

**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.

|  |
| --- |
| From the figure, we can see that model 2 has a better fit than model 1, as the line from model 2 is closer to the data points as compared to model 1 (further away). For model 1, the line does not fit very well to data points, and does not accurately capture the relationship between inverse distance to nearest mrt and price per square meter. For model 2, there seems to be same number of data points on right and left side of line, and line is able to better represent the relationship better.  The data points seems to follow a non linear pattern. There is a concentration of houses are around the left region, where they have around 2-3 inverse distance to nearest MRT (km-1), and houses cost around 4-7.5 thousand per square meter.  We note that simple linear regression are still not able to capture the non linear nature/relationship of the data points formed by inverse distance to nearest mrt against price per square meter. |

(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.

|  |
| --- |
| a cannot be interpreted as such, this is because it is impossible for predictor variable to be 0, that is impossible and not meaningful for inverse distance to nearest MRT (1/distance to mrt) to be 0. No real value of nearest MRT distance will result in inverse distance to nearest MRT to be 0. |

**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 *price per square metre* (*y*) using *inverse* *distance to the nearest MRT station* (*x*) and *inverse age of flat* (*w*) as the predictors.

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

|  |
| --- |
| Firstly, we need to perform feature **scaling** (standardization), where features should be on similar scale. Features with different scales will lead to more iterations for convergence. For instance, inverse age of flat is of a very different scale from inverse distance to nearest MRT. By performing scaling for features to take on a -1 <= x <=1 range, gradient descent algorithm would converge to minimum faster. This is because there would be a more direct and efficient path to minimum point, and fewer iterations are required.  We also need to find the partial derivative of cost function with respect to a,b and c, which is needed when we update a,b and c values respectively for each iteration (updated\_value = original\_value - Learningrate \* partial derivative)  **Partial Derivatives and cost function:**    where m is the number of data instances, is the actual response value (price per square meter) for the ith data point, a is the intercept, the predictors value for the ith  data point. B and c are slopes |

(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.

|  |
| --- |
| a = 4.4215, b= 0.176, c = 16.8133 (NOTE THAT SCALING IS DONE and a, b ,c applies to UNSCALED features.)  import pandas as pd  df = pd.read\_csv('housing\_price.csv')  import numpy as np  from sklearn.preprocessing import StandardScaler  predictor1 = 'inverse distance to the nearest MRT station (km-1)'  predictor2 = 'inverse age of flat (year-1)'  price = 'price per square metre ($ in thousands)'  df\_copy = df.copy()  sc = StandardScaler().fit(df[[ predictor1  , predictor2]])  df[[ predictor1  , predictor2]] = sc.transform(df[[ predictor1  , predictor2]])  def costFunction(df, a,b,c, pred1 = predictor1,pred2 = predictor2, target = price):      squarederrors = (df[target] - a -  b\*df[pred1] - c\*df[pred2])\*\*2      return sum(squarederrors) / len(df)    def partial\_diff\_a(df, a,b,c, pred1 = predictor1,pred2 = predictor2, target = price  ):      return (-2 \* sum((df[target] - a -  b\*df[pred1] - c\*df[pred2])))/len(df)  def partial\_diff\_b(df, a,b,c, pred1 = predictor1,pred2 = predictor2, target = price ):      return (-2 \* sum(df[pred1]\*(df[target] - a - b\*df[pred1]  - c\*df[pred2] )))/len(df)  def partial\_diff\_c(df, a,b,c, pred1 = predictor1,pred2 = predictor2, target = price ):      return (-2 \* sum(df[pred2]\*(df[target] - a  - b\*df[pred1]- c\*df[pred2])))/len(df)  a,b,c = 2,2,2  # Starting value of a b c  rate = 0.5  # Set learning rate  epsilon = 0.0001 # 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  mse = costFunction(df,a,b,c)  while  iter <= max\_iter  and diff > epsilon:      b\_new = b - rate \* partial\_diff\_b(df,a,b, c )      a\_new = a - rate \* partial\_diff\_a(df,a,b, c  )      c\_new = c - rate \* partial\_diff\_c(df,a,b, c  )      mse\_new =  costFunction(df,a\_new , b\_new, c\_new )      print("Iteration ", iter  , 'a-value is :' , a\_new, ": b-value is: ", b\_new , 'c-value is'  , c\_new,"E(a,b,c) is: ", mse\_new)      diff = abs(mse\_new -  mse)      iter = iter + 1      mse = mse\_new      b = b\_new      a = a\_new      c = c\_new  print("The local minimum for scaled a: " , a , 'scaled b:', b , 'scaled c:' , c )  # To find original coefficients and intercepts  unscaled\_b = b /df\_copy[predictor1].std()  unscaled\_c = c /df\_copy[predictor2].std()  unscaled\_a = a + (b \* (- df\_copy[predictor1].mean() )/df\_copy[predictor1].std() ) + (c \* (- df\_copy[predictor2].mean())/df\_copy[predictor2].std())  print("The local minimum for original a: " , unscaled\_a , 'original b:', unscaled\_b , 'original c:' , unscaled\_c ) |

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

|  |
| --- |
| **EPSILON**   * Chosen to be value of 0.0001, to get optimal and accurate values for a, b and c, while minimising cost function.   **LEARNING RATE (α)**  If alpha is set to 0.01, convergence occurs only after 231 iterations, which is too slow. Changes in cost function for several consecutive iterations is also not very significant, suggesting that learning rate is too small.    If alpha is set to 0.8, we can see from figure below that from the first few iterations, cost function does not decrease on every iteration but increases rapidly. This leads to divergent outcomes, diverging away from optimal solution, suggesting that learning rate is too high, and gradient descent does not reach convergence, missing/overshoot the minimum. Ideally, we want the cost function to decrease every iteration.    Above evaluation showed that the optimal learning rate is between 0.01 and 0.8. **Adjusting to 0.5 through trial and error,** we can see from figure below that it only takes 20 iterations to converge which is pretty quick unlike the case for α = 0.01 which takes 231 iterations. For alpha = 0.5 the cost function have a somewhat **significant decrease for every iteration**. Unlike the case for alpha = 0.8, cost function DOES NOT increase for every iteration. Figure below shows cost function decreases from 2.57 to 0.1563 in 20 iterations.    We can conclude that gradient descent algorithm has converged to the optimal solution of the cost function  **STARTING A,B,C values**   * a,b and c is chosen to start at 2. (random initialization) |

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

|  |
| --- |
| Final MODEL 3 equation is:  **(Performed scaling, and these are the coefficients and intercept for the original features)**    **(working to find coefficients for original features)**  Minimum value of Error function is: 0.15628858937471693 ≈ 0.156 (3sf)  Number of iterations ran to reach convergence: 20 |

**Conclusion** (10 marks)

(a) David used gradient descent algorithm to find the 3 models. Next, he computed the predicted housing prices 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 housing price in Model 1 from the actual housing price is the smallest, compared to the other 2 models. Is this possible, assuming he has done his gradient descent algorithm correctly? Explain.

|  |
| --- |
| Yes, it is possible, assuming he has done gradient descent correctly.  **Parameters can be the reason why for 1 data point, model 1 has the lowest error**  For example, if David were to select the parameters to be the following: epsilon: 0.012, learning rate: 0.002, starting x: 2, max\_iter:100 for MODEL 1, and the same parameters for model 2 and 3 as my implementation, he would get b/gradient to be as 0.8555091 From the figure below, we see convergence has occurred after 14 iterations. Equation for model 1 would be: = 0.8555091    **Dataframe below that shows the house predictions for each model compared to the actual price per square meter (thousands).** We can then generate the predictions for model 1, in comparison to model 2 and 3 for all 30 houses. We can compare the model predictions against actual price. From the dataframe below we can see that for data point 9, model 1 has the lowest error, given the smallest difference between predicted value and actual value.    Absolute error for model 1 for datapoint 9: 0.001, Abs error for model 2: 0.349, abs error for model 3: 0.061. This shows that it is possible for 1 data point to have the least error for model 1. |

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

|  |
| --- |
| Model 1 - MSE: 19.467  Model 2 – MSE: 0.4696  Model 3 – MSE: 0.156  The minimum cost function (aka mean squared error) for each model can be compared directly to see which models performs the best by having the least error for predictions.  I will use model 3, as it has the lowest mean squared error overall, when we compare the actual price per square meter with the predicted price per square meter for each model for 30 houses. Model 1 has the highest average squared error, suggesting that it performs the worst, and is the least optimal solution. Model 2 acts as an improvement as compared to model 1, by accounting for the intercept, having a mean squared error of 0.4696. However, model 3 further improves price prediction by using more information (inverge age of house), this could provide information about the quality of state and level of newness of the house. Hence with more predictors that provide more information towards target variable, predictions would be more accurate. Hence model 3 is used with lowest mean squared error. |