**Logo

Description automatically generated**

**San Francisco Bay University**

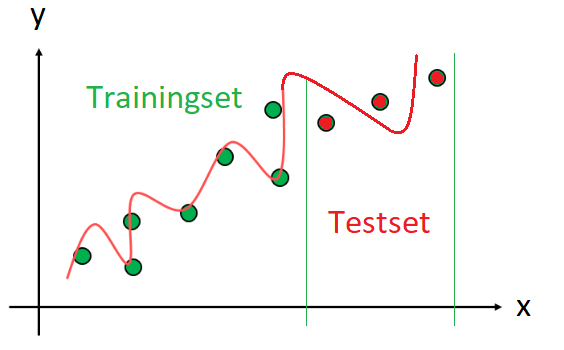
**CS483 - Fundamentals of Artificial Intelligence**

**2022 Summer Midterm Exam**

**Student Name: Mahmud Hamed Omer Student ID:19660**

**Instruction**

1. **Put your answer right after each question in the answer sheet**
2. **Make copy & paste for your program in text mode, NOT image onto the answer sheet.**
3. **Excel is preferred for hand calculation**
4. If we want to fit **trainingset** (X, y) with **single** feature in a squiggle curve as follows, please create **hypothesis/loss/cost** functions and **partial derivative** expression *vs* each coefficient for future gradient descent algorithm implementation first. But if you find that this hypothesis function will generate quite large errors for the testset as follows after modeling, how to tweak it?



***\*Hint:*** *high order polynomial hypothesis function is one of options. You should make decision about what the highest order is based on your fitting observation.*

**Solution**

To obtain a good fitting or at least closer to the given fitting curve, 3rd order polynomial function is used. This is chosen to avoid overfitting and code complexity if higher is chosen, and under fitting for linear hypothesis function

**Hypothesis function**: +

; where

**Loss function**: =

**Cost function**: =

; where N = 7 (training data set )

**Minimum values of cost function**:

= \*1

= \*

= \*

= \*

Obtain the **hypothesis function coefficients** using the **gradient descent method**

*However, the function will generate large for the testing set, because the testing set data are linear, hence we need to reduce the degree of the hypothesis function so that it can fit well*

1. Write python program to make feature scaling for "***Alcohol***" and "***Malic acid***" in the following dataset

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Alcohol | Malic acid | Class label |
| 0 | 14.23 | 1.71 | 0 |
| 1 | 13.2 | 1.78 | 1 |
| 2 | 13.16 | 2.36 | 1 |
| 3 | 14.37 | 1.95 | 0 |
| 4 | 13.24 | 2.59 | 0 |

**Solution:**

**Code:**

|  |
| --- |
| # A = "Alcohol" , M =  "Malic acid"  A = [ 14.23, 13.2, 13.16, 14.37, 13.24]  M = [1.71, 1.78, 2.36, 1.95, 2.59]  print("---------------- Feature scaling for Alcohol---------------")  print("max: ",max(A))  print("min: ",min(A))  print("range: ", max(A)-min(A))  sum=0  rng=max(A)-min(A)  for i in range (len(A)):      sum+=A[i]  print("sum: ",sum)  average = sum/len(A)  print("mean = " ,average)  As = [0]\*len(A)    # A scalled  for i in range(len(A)):      devation = float(A[i]-average)      As[i] = devation/rng  As= ["%.3f" % elem for elem in As]  print("A = ",As)  print("---------------- Feature scaling for Malic Acid ---------------")  print("max: ",max(M))  print("min: ",min(M))  print("range: ", max(M)-min(M))  sum=0  rng=max(M)-min(M)  for i in range (len(M)):      sum+=M[i]  print("sum: ",sum)  average = sum/len(M)  print("mean = " ,average)  Ms = [0]\*len(A)    # A scalled  for i in range(len(M)):      devation = float(M[i]-average)      Ms[i] = devation/rng  Ms= ["%.3f" % elem for elem in Ms]  print("M = ",Ms) |

Result:

|  |
| --- |
|  |

1. Given the dataset as below, write python program or hand calculation to find in linear regression hypothesis function by **normal** equation method

|  |  |
| --- | --- |
| X | y |
| 1 | 7 |
| 2 | 9 |
| 3 | 12 |
| 4 | 15 |
| 5 | 16 |

Solution:

Code

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from matplotlib import colors  #initila data  x\_init = [ 1, 2, 3, 4, 5]  y\_init = [ 7, 9, 12, 15, 16]  # Using Normal Equations  y = np.array([[7 ],[9] ,[12] ,[15] , [16]])  x =np.array([[1, 1, 1, 1, 1],[1, 2, 3, 4, 5]])  X = x.transpose()  X\_trans = X.transpose()  [t\_0, t\_1]= np.linalg.inv(X\_trans.dot(X)).dot(X\_trans).dot(y)  print(" Theta\_0 = ", t\_0, " \n Theta\_1 = ", t\_1)  # predicted values  y\_pred = t\_0 + t\_1\*x\_init  # compare results  plt.scatter(x\_init,y\_init, color = 'b')  plt.plot(x\_init, y\_pred, color = 'r')  plt.xlabel('X Label')  plt.ylabel('Y Label')  plt.show() |

Result

|  |
| --- |
|  |

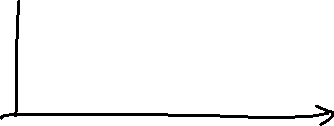
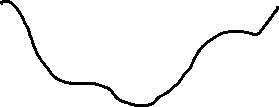
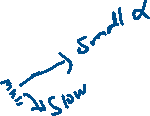
1. In the process of applying gradient descent algorithm to find max value for each coefficient in hypothesis function, appropriate learning rate α is very important. Please discuss about how it makes the impacts on your training result

**Solution:**

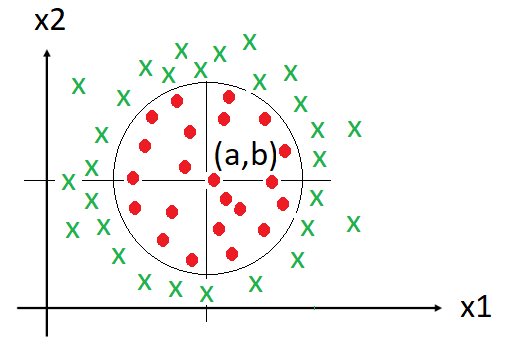
The choice of appropriate learning rate (α) is important because if we use large alpha value, then we can easily miss the optimal solution, and if very small learning rate is used, the program takes long time to converge (which may cost more time, money and energy for large system applications).



e.g.



1. A training set with *2* features (, ) and binary class labeled in "o" and "x" as *0* and *1* respectively is visualized as follows. In terms of our observation, circle decision boundary in **logistic** regression is a good option for binary classification. Please buildup hypothesis/cost function and partial derivative expression vs each coefficient for gradient descent algorithm application



***\*Hint:*** *circle equation is*

***Solution:***

*Let’s write the boundary equation in a standard format that can be used in hypothesis function, which is a sigmoid function in this case*

*Boundary Function = ( =( -2\*a\* + ) + (( -2\*b\* + )*

*= + (-2\*a\* + +*

*Let , ,*

*,*

*Hence, the boundary function can be written in terms of theta values as follows*

*Boundary Function = + + +*

**Hypothesis function:**

**Loss function**:

=

Where

**Cost function:**

=

**Partial derivative function:**

=

= \*

= \*

= \*

but we know , similarly

= \*

but we know

**Then we can use the gradient decent algorithm:**

# Again, we know

# and the same way

1. KNN algorithm is one of supervised learning algorithms. If *K=1*, what is error rate in given "**Testset**" after training based on "**Trainingset**" from "iris" dataset as follows?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Trainingset** |  |  |  |  |
| **sepal length** | **sepal  width** | **petal length** | **petal width** | **Species (label)** |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3 | 1.4 | 0.2 | setosa |
| 6.6 | 2.9 | 4.6 | 1.3 | versicolor |
| 5.2 | 2.7 | 3.9 | 1.4 | versicolor |
| 6.9 | 3.1 | 5.1 | 2.3 | virginica |
| 5.8 | 2.7 | 5.1 | 1.9 | virginica |
|  |  |  |  |  |
| **Testset** |  |  |  |  |
| **sepal length** | **sepal  width** | **petal length** | **petal width** | **Species**  **(label)** |
| 7 | 3.2 | 4.7 | 1.4 | versicolor |
| 5 | 3.3 | 1.4 | 0.2 | setosa |
| 5.9 | 3 | 5.1 | 1.8 | virginica |

**Solution:**

**Data in csv format ( csv file to be uploaded to collab)**

|  |
| --- |
| **5.1, 3.5, 1.4, 0.2, setosa**  **4.9, 3, 1.4, 0.2, setosa**  **6.6, 2.9, 4.6, 1.3, versicolor**  **5.2, 2.7, 3.9, 1.4, versicolor**  **6.9, 3.1, 5.1, 2.3, virginica**  **5.8, 2.7, 5.1, 1.9, virginica**  **7, 3.2, 4.7, 1.4, versicolor**  **5, 3.3, 1.4, 0.2, setosa**  **5.9, 3, 5.1, 1.8, virginica** |

**Then the following code , which was used in the class is used to check the error for k = 1**

|  |
| --- |
| from sklearn.metrics.pairwise import euclidean\_distances  # Q6  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.neighbors import KNeighborsClassifier  from sklearn.metrics import f1\_score  from sklearn.metrics  import accuracy\_score  dataset = pd.read\_csv('Q6.csv')  print(len(dataset))  print(dataset.head())  # Replace Zeros  # zero\_not\_accepted = ['SepalLength', 'SepalWidth', 'PetalLength','PetalWidth','SpeciesLabel']  # for column in zero\_not\_accepted:  #     dataset[column] = dataset[column].replace[0,np.NaN]  #     mean = int(dataset[column].mean(skipna= True))  #     dataset[column] = dataset[column].replace(np.NaN,mean)  # dive the data into train and test sets  X = dataset.iloc[:,0:4]  y = dataset.iloc[:,4]  X\_train, X\_test,y\_train, y\_test = train\_test\_split(X,y,random\_state=0,test\_size=0.33 )  # feature scaling  sc\_X = StandardScaler()  X\_train = sc\_X.fit\_transform(X\_train)  X\_test = sc\_X.transform(X\_test)  # define the model: Init K-NN  classifier = KNeighborsClassifier(n\_neighbors=1 , p = 2, metric= "euclidean")  classifier.fit(X\_train, y\_train)  # predict the test results  y\_pred = classifier.predict(X\_test)  y\_pred  # evaluate Model  # confusion metrix  # cm = confusion\_matrix(y\_test, y\_pred)  # print(cm)  # f1 score  # f1\_sc = f1\_score(y\_test, y\_pred)  # print(f1\_sc)  # accuracy score  ac\_sc = accuracy\_score(y\_test, y\_pred)  print("Error = ", 1 - ac\_sc) |

1. If ***K = 2***, please classify the follows points (, ) in K-Means algorithm to different classes and calculate **total** "within cluster sum of square" (WCSS) value in either python program or hand calculation

|  |  |  |
| --- | --- | --- |
| ID | X1 | X2 |
| A1 | 2 | 10 |
| A2 | 8 | 4 |
| A3 | 5 | 8 |
| A4 | 6 | 4 |
| A5 | 1 | 2 |

Solution:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | **Cluster 1** | **Cluster 2** |
|  | **Pnts(x1)** | **Pnts(X2)** | **Dist to A1(2,10)** | **Dist to A4(5,8)** | **Cluster** |
| A1 | 2 | 10 | 0 | 10 | Cluster 1 |
| A2 | 8 | 4 | 12 | 2 | Cluster 2 |
| A3 | 5 | 8 | 5 | 5 | Cluster 1 |
| A4 | 6 | 4 | 10 | 0 | Cluster 2 |
| A5 | 1 | 2 | 9 | 7 | Cluster 2 |

|  |  |
| --- | --- |
| **Cluster 1** | **Cluster 2** |
| A1(2, 10) | A2(8, 4) |
| A3(5, 8) | A4(6,4) |
|  | A5(1,2) |

Iteration 1:

|  |  |  |
| --- | --- | --- |
|  | **Pnts(x1)** | **Pnts(X2)** |
| A1 | 2 | 10 |
| A3 | 5 | 8 |
| Mean | 3.5 | 9 |

|  |  |  |
| --- | --- | --- |
|  | **Pnts(x1)** | **Pnts(X2)** |
| A2 | 8 | 4 |
| A4 | 6 | 4 |
| A5 | 1 | 2 |
| mean | 5 | 3.33 |

Using K-means clustering from sklearn module

|  |
| --- |
| import numpy as np  import pandas as pd  from sklearn.cluster import KMeans  from sklearn.preprocessing import MinMaxScaler  #%matplotlib inline  df = pd.read\_csv("/content/Q6.csv")  df.head()  #show data graph  from matplotlib import pyplot as plt1  plt1.scatter(df.X1,df.X2)  plt.show()  # apply feature scaling  # scaler = MinMaxScaler()  # df['X2'] = scaler.fit\_transform(df[['X2']])  # df['X1'] = scaler.fit\_transform(df[['X1']])  # df  # apply KM clustering    km = KMeans(n\_clusters = 2)  y\_predicted  = km.fit\_predict(df[["X1", "X2"]])  df["cluster"] = y\_predicted  km.cluster\_centers\_  #  plot clustered data  from matplotlib import pyplot as plt  df1 = df[df.cluster == 0]  df2 = df[df.cluster == 1]  plt.scatter(df1.X1, df1.X2, color = "blue")  plt.scatter(df2.X1, df2.X2, color = "Red")  plt.xlabel("X1")  plt.ylabel("X2")  plt.legend  plt.show() |