Regression Models

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
import math
#from sklearn.linear_model import LinearRegression
```

This is a regression problem in which the goal is to use meteorological and other data to predict the burnt area of forest fires in the northeast region of Portugal.

Separating training data

In [2]:

```
data = pd.read_csv('Forest_Fires_1A.csv')
#df = pd.DataFrame({'SEED':data})
#np.random.seed(5)
df = pd.DataFrame(data)
n = 80
df_1 = df.head(int(len(df)*(n/100)))
l = len(df_1)
x = df_1.iloc[0:,0:12]
x['13'] = 1
x = np.array(x)
area_train = df_1['area']
area_train = np.array(area_train)
x_uni_closed = df_1.iloc[0:,9:10]
x_uni_closed['1'] = 1
x_uni_closed = np.array(x_uni_closed)
```

Separating test data

In [3]:

```
n = 20
df_2 = df.tail(int(len(df)*(n/100)))
x_test = df_2.iloc[0:,0:12]
x_test['13'] = 1
x_test = np.array(x_test)
area_test = df_2['area']
RH_test = df_2['RH']
```

Linear Regression(Closed form) Multivariate

In multivariate models we take more than one variate.
Generally in Linear Regression we try to fit a straight line.
Data : {X(train), y}.
Model : y_pred = w.T @ X (where w is weights of all variates).
Loss function : summation(y_pred - y)^2.
We have to choose weights such that loss function is minimised.
To find weights,
In closed form solution we differentiate loss function with all weights and make them equal to 0.
By soloving equation the equation we get: W = (X.T @ X)^(-1) @ (X.T@Y)

In [4]:

In [4]:

```
w = np.linalg.inv(x.T@x)@(x.T@area_train)
print(w)
```

```
[ 0.93876828   0.03496786   1.20255328   0.04304955  -0.05813767
  0.13194779  -0.01153822  -0.66874919   0.34491199  -0.21315533
  0.72024577 -10.08203177   3.16205712]
```

Now we got all weights.
We put them in equation and predict Y for test data

In [5]:

```
Area_predict_closed = x_test @ w
print(Area_predict_closed)
```

```
[ 25.99337428  21.78616853  11.3726493   18.41538758  16.74864365
  24.38718899  19.64622927  40.85379613  10.79495466  28.61102423
  17.70811684  23.48059794  19.31288933  18.02484102  15.878219
  41.29932851  12.61375019  23.70342482  13.66236478  23.39550621
   7.75451213  23.56149968  16.36435283  17.40391808   9.22016572
  35.23894844  24.75216898  -3.41624611  -5.65894846  32.45761146
  -5.94406038  11.93180321   6.07113987  29.33363886  14.82983909
  13.25994613  -3.40868928  18.91466134  22.10390884  18.12335887
   7.96041391  16.18804388  12.15869653  19.75392178  23.12444533
  14.08634425  19.75392178  28.77155858  -3.80529139  -3.25024777
  -7.06527579   7.2828938    5.64830297  10.20847222   1.60511408
   0.73251766  10.68574719  10.78969051  12.67735474  15.41218665
  13.20792591  12.19763484   8.6580605    1.257904    14.36708789
  23.30108689  19.32500267   6.64416702  16.51219627  10.98690051
   4.83515825   7.38643427   8.15346629  -0.20545022  -1.51503526
   8.55229877  16.42847828  12.81764296  12.35782882  17.32761629
   9.81850221  14.96543862  15.50718562  19.27664852 -52.68126515
   2.3595399    1.38580377   1.40430165   5.41983022   7.37208067
   9.49060114   5.61570954  12.46852869  11.4598247   -3.99926546
  10.42292772  10.07343503   6.85292435  -3.10492126   2.20885827
   7.41432617  14.44819622]
```

We got Y_predict, now to see efficiency of the model we calculate the mean square error which is same as loss function.
summation(y_pred - y)^2

In [6]:

```
Error_closed = Area_predict_closed - area_test
Error_closed = Error_closed@Error_closed.T
print(Error_closed)
```

```
659754.2977067346
```

We got error as 659754.2977067346.
Observation : Results are better for large datasets with diversity

Linear Regression (Closed form) Univariate

In univariate models there is only one variate.
Generally in Linear Regression We try to fit a straight line of the form y = ax + b.
Data : {x(train), y}.
Model : y_pred = ax + b (where a,b are weights of all variates).
Loss function : summation(y_pred - y)^2.
We have to choose weights such that loss function is minimised.
To find weights, In closed form solution we differentiate loss function with weights and make them equal to 0.
By soloving equation the equation we get: W = (X.T @ X)^(-1) @ (X.T@Y).
Here for univariate Iam taking RH(Relative Humidity)

In [7]:

```
w_uni = np.linalg.inv(x_uni_closed.T@x_uni_closed)@(x_uni_closed.T@area_train)
print(w_uni)
```

```
[-0.25478819 22.41071466]
```

Now we got a,b.
We put them in equation and predict Y for test data

In [8]:

```
Area_predict_uni_closed = w_uni[0]*RH_test + w_uni[1]
print(Area_predict_uni_closed)
```

```
411    15.531434
412    12.473975
413    16.295798
414    11.709611
415    13.238340
         ...
508    14.257493
509     4.320753
510     4.575541
511    11.709611
512    14.512281
Name: RH, Length: 102, dtype: float64
```

We got Y_predict, now to see efficiency of the model we calculate the mean square error which is same as loss function.
summation(y_pred - y)^2

In [9]:

```
Error_uni_closed = Area_predict_uni_closed - area_test
Error_uni_closed = Error_uni_closed@Error_uni_closed.T
print(Error_uni_closed)
```

```
663759.9746236483
```

We got error as 663759.9746236483.
Observation : Error in univariate is more than error in Multivariate. So we can say that RH is not very good variate to predict area burnt and more variates give good results

Linear Regression (Gradient descent) Multivariate

In multivariate models we take more than one variate.
Generally in Linear Regression we try to fit a straight line.
Data : {X(train), y}.
Model : y_pred = w.T @ X (where w is weights of all variates).
Loss function : summation(y_pred - y)^2.
We have to choose weights such that loss function is minimised.
The difference between closed and gradient is the optimization method which we use for finding weights.
Gradient descent : Θ(new) = Θ(old) - learingrate * d/dΘ(loss function)

In [10]:

```
def gradient_descent(data, learning_rate, weight, num_iters, n):##function to calculate w
eights with gradient descent
    for i in range(num_iters):
        gra = np.zeros(n)
        for j in range(len(data)):
            for k in range(n):
                gra[k] += ((data[j]*weight).sum() - area_train[j])*data[j][k]
        weight = weight - learning_rate*gra
```

```
    return weight
```

Training the data

In [11]:

```
learning_rate = 0.00000001
num_iters = 500
weight = np.zeros(13)
weight = gradient_descent(x, learning_rate, weight, num_iters, 13)
print(weight)
```

```
[ 1.03141470e-02  5.01684683e-03  3.71995658e-02 -9.91666755e-04
  4.76843479e-02  6.93020247e-02  4.88981995e-03 -1.77573703e-02
  2.83275199e-02 -7.94582093e-02  3.19038045e-03 -8.89734442e-05
  5.39698543e-04]
```

Now we got all weights.
We put them in equation and predict Y for test data

In [12]:

```
Area_predict_gradient = x_test @ weight
print(Area_predict_gradient)
```

```
[21.89810212 16.36486745  4.4596098  19.03538055 17.73011224 18.65847067
 22.1305671  25.26425745 11.63840097 22.28993101 21.35625916 21.43821143
 20.13320904 21.22069065 17.09407836 25.69645599 17.07241109 20.92782543
 19.55153651 24.01970229  8.79952091 20.01217958 20.18185105 20.38274401
 11.48073844 24.79753035 20.45369825  1.37464074  5.8880164  24.77160475
  2.36348199 16.68231529  3.41782419 22.65706115 15.63728088 19.56571366
 14.26821678 19.78857134 22.85188992 21.27749928 15.01445934 21.02186884
 17.27373076 19.44533743 22.08293129 19.3784357  19.44533743 22.22925168
 -2.23951848 -1.83595186 -0.33079544  2.33763234  3.03726462  4.42978104
  3.12830477  3.69874868  3.99072358  6.36987906  6.68368886 11.23652168
 12.05504095 10.91433432 10.71399496  7.58780828  9.85525248 14.1458794
 14.30385836 13.7645467  15.07312085 15.2201568  15.18892595 14.36561373
 15.13727689 12.00147787 13.04733656 15.27185873 17.26739575 17.26274329
 16.9723599  17.47089659 17.16303396 18.65804094 19.02159611 19.16693631
 16.073781   15.76922697 15.75389597 15.37464467 15.71283576 15.79940716
 17.14516921 14.26859993 17.02560125 17.00525341 14.54946228 15.19087307
  9.20385302  9.38592052  6.11419672  6.22826775 14.8914508   2.79375225]
```

We got Y_predict, now to see efficiency of the model we calculate the mean square error which is same as loss function.
summation(y_pred - y)^2

In [13]:

```
Error_gradient = Area_predict_gradient - area_test
Error_gradient = Error_gradient@Error_gradient.T
print(Error_gradient)
```

```
662333.0766500426
```

We got error as 662333.0766500426
Observation : Results are better for large datasets with diversity

Linear Regression (Gradient descent) Univariate

In univariate models there is only one variate.
Generally in Linear Regression We try to fit a straight line of the form y = ax + b.
Data : {x(train), y}.
Model : y_pred = ax + b (where a,b are weights of all variates).
Loss function : summation(y_pred - y)^2.

We have to choose weights such that loss function is minimised.
In gradient descent our minimization method is :
Gradient descent : Θ(new) = Θ(old) - learingrate * d/dΘ(loss function)

In [14]:

```
learning_rate_uni = 0.00000001
num_iters_uni = 1000
weight_uni = np.zeros(2)
weight_uni = gradient_descent(x_uni_closed, learning_rate_uni, weight_uni, num_iters_uni
, 2)
print(weight_uni)
```

```
[0.19953193 0.01485845]
```

Now we got all weights.
We put them in equation and predict Y for test data

In [15]:

```
Area_predict_gradient_uni =  weight_uni[0]*RH_test + weight_uni[1]
print(Area_predict_gradient_uni)
```

```
411     5.402221
412     7.796604
413     4.803625
414     8.395199
415     7.198008
          ...
508     6.399880
509    14.181625
510    13.982093
511     8.395199
512     6.200348
Name: RH, Length: 102, dtype: float64
```

We got Y_predict, now to see efficiency of the model we calculate the mean square error which is same as loss function.
summation(y_pred - y)^2

In [16]:

```
Error_gradient_uni = Area_predict_gradient_uni - area_test
Error_gradient_uni = Error_gradient_uni@Error_gradient_uni.T
print(Error_gradient_uni)
```

```
679497.2457544276
```

Our error is 679497.2457544276.
Observation : Error in univariate is more than multivariate which means our variate RH is not a good variate and we get better results for more variates and diverse data

Linear Regression (Newton's Method) Multivariate

In multivariate models we take more than one variate.
Generally in Linear Regression we try to fit a straight line.
Data : {X(train), y}.
Model : y_pred = softmax(w.T @ X) (where w is weights of all variates).
Loss function : summation(y_pred - y)^2.
We have to choose weights such that loss function is minimised.
The difference between closed and gradient and newton's method is the optimization method which we use for finding weights.
Θ(new) = Θ(old) - H^(-1)@G
G is gradient.

G = X.T@(Y_pred - Y)
H is Hessian matrix
H = X.T @ S @ X
S = Y_pred(1 - Y_pred)

In [17]:

```
def gradient_descent_newton(data, weight, num_iters, n, correction):#function to calculat
e weights from newtons optimization
    for i in range(num_iters):
        gra = np.zeros(n)
        s = np.zeros(len(data))
        for j in range(len(data)):
            for k in range(n):
                a = (data[j]*weight).sum()
                b = (area_train[j] - a)
                gra[k] += (a - area_train[j])*data[j][k]
            s[j] = b
        S = np.diag(s)
        h = data.T @ S @ data
        weight = weight - np.linalg.inv(h - correction * np.identity(n))@gra
    return weight
```

Training the data

In [18]:

```
num_iters_n = 50
weight_n = np.zeros(13)
weight_n = gradient_descent_newton(x, weight_n, num_iters_n, 13, 0.0001)
print(weight_n)
```

```
[ 1.81896469e-06  2.93257981e-06 -6.02911777e-08  1.13081803e-06
 -2.61506007e-07 -8.74825017e-08  4.15367477e-09  3.07652225e-07
  1.19031918e-06  4.27617629e-07 -3.96743786e-07 -2.52540665e-05
  4.99999667e+01]
```

Now we got all weights.
We put them in equation and predict Y for test data

In [19]:

```
Area_predict_gradient_n = x_test @ weight_n
print(Area_predict_gradient_n)
```

```
[50.00000857 50.00000505 50.00000025 49.99999427 49.99999673 50.00002055
 49.99999625 50.00000557 49.99999797 50.00000573 49.9999874  50.00001005
 49.99999271 49.99998813 49.9999988  49.99999296 49.99998858 50.00000828
 49.99999762 49.99998219 50.00000851 50.00000939 50.0000087  49.9999972
 49.99999165 49.99998636 50.00000508 50.00000866 49.99999778 49.99998384
 50.00002174 50.00001082 49.99998986 49.99999935 50.00000668 50.00000649
 49.99999675 50.00000008 49.99999406 49.99999045 49.99999423 49.99998983
 49.99999477 50.0000098  49.99999729 49.99999595 50.0000098  49.99998736
 50.00001641 50.0000123  49.99998814 50.00000028 49.99999618 49.99999944
 49.99999531 49.99999992 49.99999564 50.00001095 50.00002094 50.00000349
 50.0000014  50.0000036  50.0000075  50.00002211 50.00001267 50.0000294
 50.00000127 50.00000043 50.00001904 50.00000739 50.00000244 50.00000842
 50.00000838 50.00001247 50.00001197 50.00000985 50.00000441 49.9999937
 49.99998781 50.00001248 50.00000922 50.00000581 50.00000481 50.00001377
 49.99986177 50.00000201 49.99999726 49.99999961 50.00000654 50.00000646
 49.999993   49.9999911  49.99999585 49.99998817 49.99997148 50.00000516
 50.00002621 50.00000886 50.00001657 50.00002405 49.99999786 49.99999677]
```

We got Y_predict, now to see efficiency of the model we calculate the mean square error which is same as loss function.
summation(y_pred - y)^2

In [20]:

```
Error_gradient_n = Area_predict_gradient_n - area_test
Error_gradient_n = Error_gradient_n@Error_gradient_n.T
print(Error_gradient_n)
```

```
757988.0482076426
```

Our error is 757988.0482076426.
Observation : We get better results for large and diverse datasets

Linear Regression (Newton's Method) Univariate

In univariate models there is only one variate.
Generally in Linear Regression We try to fit a straight line of the form y = ax + b.
Data : {x(train), y}.
Model : y_pred = softmax(ax + b) (where a,b are weights of all variates).
Loss function : summation(y_pred - y)^2.
We have to choose weights such that loss function is minimised.
The difference between closed and gradient and newton's method is the optimization method which we use for finding weights.
Gradient descent : Θ(new) = Θ(old) - (d/dΘ(loss function))/(d^2/dΘ(loss function))

In [21]:

```
num_iters_nu = 500
weight_nu = np.zeros(2)
weight_nu = gradient_descent_newton(x_uni_closed, weight_nu, num_iters_nu, 2, 0.0001)
print(weight_nu)
```

```
[7.51725604e-08 4.99999995e+02]
```

Now we got all weights.
We put them in equation and predict Y for test data

In [22]:

```
Area_predict_gradient_uni_n =  weight_nu[0]*RH_test + weight_nu[1]
```

We got Y_predict, now to see efficiency of the model we calculate the mean square error which is same as loss function.
summation(y_pred - y)^2

In [23]:

```
Error_gradient_nu = Area_predict_gradient_uni_n - area_test
Error_gradient_nu = Error_gradient_nu@Error_gradient_nu.T
print(Error_gradient_nu)
```

```
24231445.924455136
```

Our Error is 24231445.924455136
Observation : Error is very high so RH is not suitable variate to predict area burnt

Conclusion:

For our dataset we got better results for Linear Regression closed form (Multivariate)