rough_work_

September 22, 2024

DA5400 Foundations of Machine Learning - Assignment 1

DA24M011 - Nandhakishore C S

Rough work & calculations done for the assignment

[1]:
```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from math import *
```

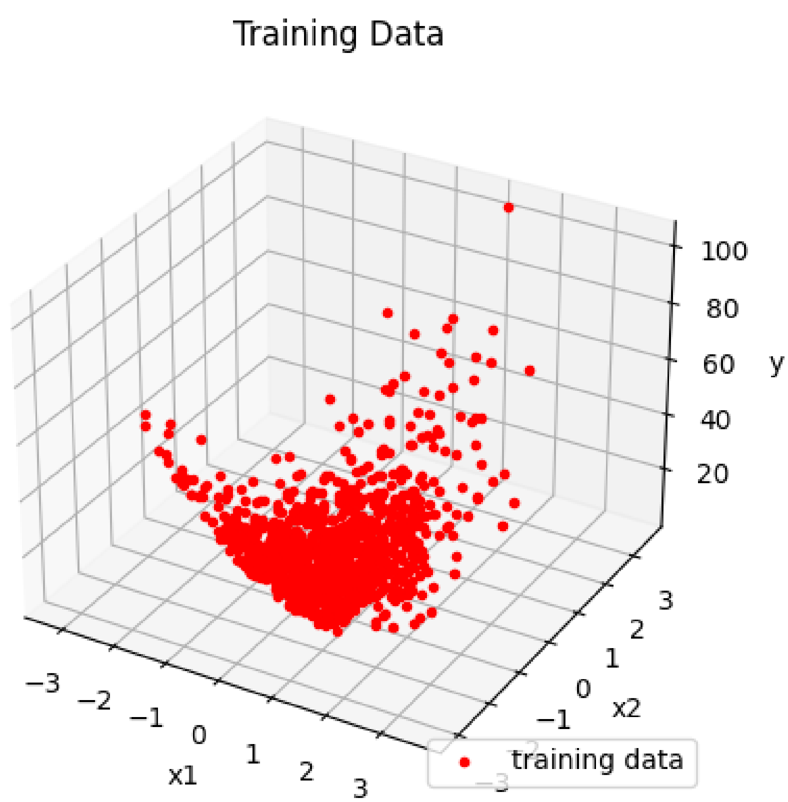
Data Processing

[2]:
```
df_train = pd.read_csv('/Users/nandhakishorecs/Documents/IITM/Jul_Nov_2024/
 ↪DA5400_Fundamentals_of_Machine_Learning/Assignment1/Dataset/
 ↪FMLA1Q1Data_train.csv', header = None)
print('\nDescription of the training data:\n',df_train.describe())
df_train.columns = ['x1', 'x2', 'y']

print('\nColumns in training dataset:\t',df_train.columns)
Y = np.array(df_train.y)
print('\nLabel shape:\t',Y.shape)
```

```
Description of the training data:
                  0            1            2
count  1000.000000  1000.000000  1000.000000
mean     -0.032632     0.036899     9.966333
std       0.998967     0.998594    11.783516
min      -3.232000    -3.072200     0.409420
25%      -0.712633    -0.620370     2.521725
50%      -0.030433     0.049701     5.644650
75%       0.610568     0.673875    12.937000
max       3.578400     3.569900   106.260000

Columns in training dataset:     Index(['x1', 'x2', 'y'], dtype='object')

Label shape:     (1000,)
```

Data Visualisation

[3]:
```
%matplotlib widget
figure = plt.figure(figsize=(5,5))
ax = plt.axes(projection = '3d')
ax.plot3D(df_train.x1, df_train.x2, df_train.y, 'r.')
plt.title('Training Data')
plt.legend(['training data'], loc = 'lower right')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
plt.show()
```

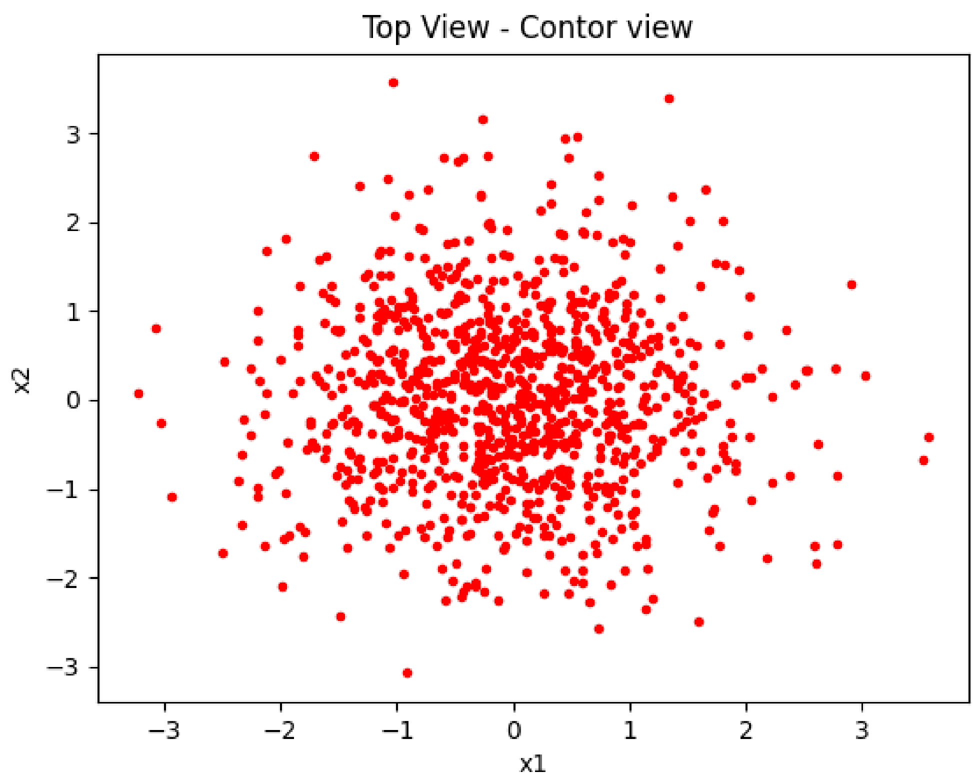


The Plot looks like where the datapoints are sampled from a eliptical paraboloid with noise

[4]:
```
plt.close()
```

[5]:

```
# Top View
plt.plot(df_train.x1, df_train.x2, 'r.')
plt.title('Top View - Contor view ')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

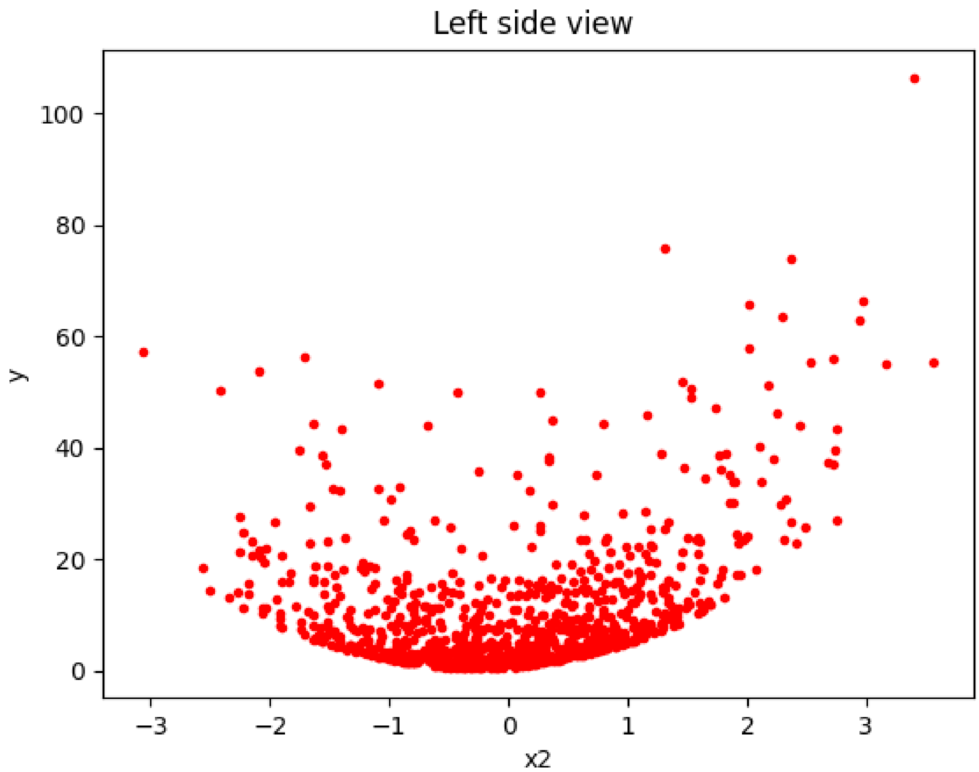


[6]:

```
plt.close()
```

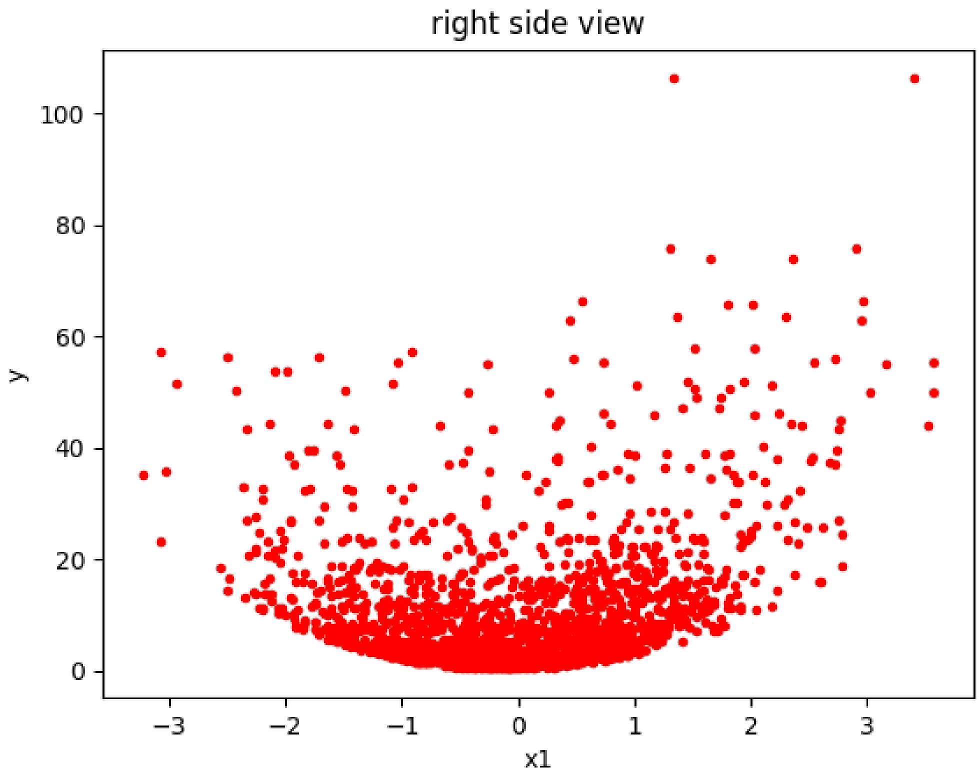
[7]:

```
# Left Side view
plt.title('Left side view ')
plt.xlabel('x2')
plt.ylabel('y')
plt.plot(df_train.x2, df_train.y, 'r.')
plt.show()
```



[8]:
```
plt.close()
```

[9]:
```
# Front view
plt.title('right side view')
plt.plot(df_train.x1, df_train.y, 'r.')
plt.xlabel('x1')
plt.ylabel('y')
plt.plot(df_train.x2, df_train.y, 'r.')
plt.show()
```



[10]:
```
plt.close()
```

Code for Calculating Analytical, Gradient Descent, Stochastic Gradient Descent solution for linear regression

[12]:
```
from tqdm.auto import tqdm
from numpy import *

class linear_model:

        # ---------------------- Class Initialisation␣
 ↪----------------------------

        __slots__ = '_lr', '_iters', '_tolerance' ,'_w', '_b'
        def __init__(self, learning_rate = 0.01, iterations = 10_000, tolerance␣
 ↪= 1e-9) -> None:
                self._lr = learning_rate
                self._iters = iterations
                self._w = None
                self._b = None
                self._tolerance = tolerance
```

```
    # -------------------------- Analytic Solution
↪---------------------------

    def analytical_solution(self, X: np.ndarray, y: np.ndarray,
↪fit_intercept: bool = False, Lambda:float = 0) -> np.ndarray:
        if(fit_intercept == False):
            penalty = np.dot(Lambda, np.identity(X.shape[1]))
            pseudo_inverse = linalg.inv(np.dot(X.T, X) + penalty)
            self._w = np.dot(np.dot(pseudo_inverse, X.T), y)
            return self._w

        elif(fit_intercept == True):
            X_bias = np.c_[X, np.ones(X.shape[0])]
            penalty = np.dot(Lambda, np.identity(X_bias.shape[1]))
            pseudo_inverse = linalg.inv(np.dot(X_bias.T, X_bias) +
↪penalty)
            w = np.dot(np.dot(pseudo_inverse, X_bias.T), y)

            # Book keeping for maintaining values in the class
            self._w = np.empty((X.shape[1], 1))

            for i in range(X_bias.shape[1]-1):
                self._w[i] = w[i]

            self._b = w[w.shape[0]-1]

            return w

    # ---------------- Helper Function to do prediction
↪----------------------

    def predict(self, X: np.ndarray, fit_intercept:bool = False) -> np.
↪ndarray:
        if(fit_intercept == True):
            return np.dot(X, self._w) + self._b
        if(fit_intercept == False):
            return np.dot(X, self._w)

    # ------------------------- Loss Functions -----------------------------

    # Sum of Squared Error
    def SSE(self, y: np.ndarray, y_pred: np.ndarray) -> float:
        error = 0
        for i in range(0, len(y_pred)):
            error += (y[i] - y_pred[i]) ** 2
        return error
```

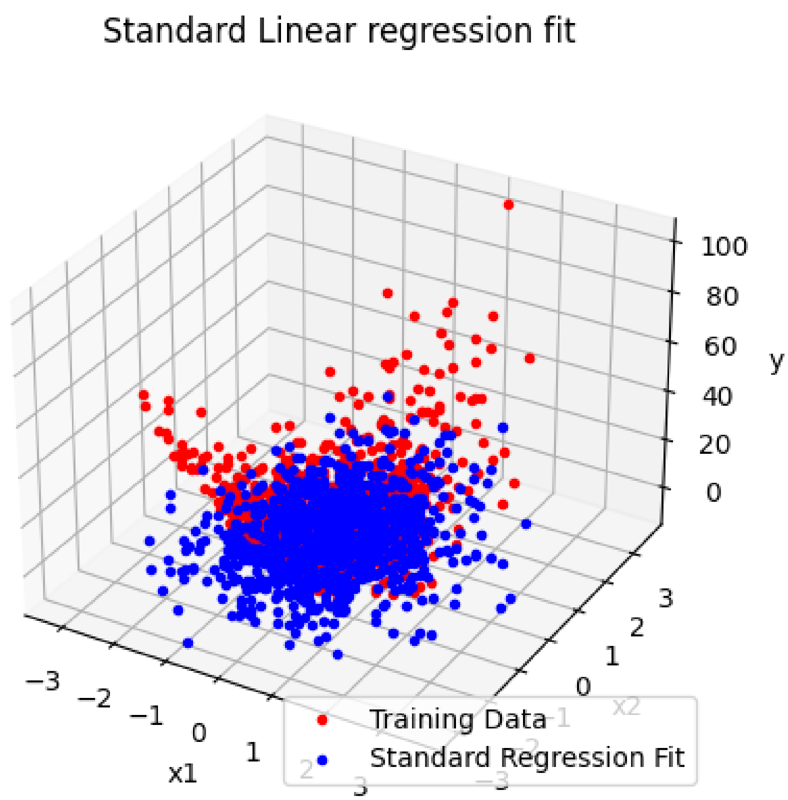
```
        # Mean of sum of Squared Error
        def MSE(self, y: np.ndarray, y_pred: np.ndarray) -> float:
                error = 0
                n_samples = len(y_pred)
                for i in range(0, len(y_pred)):
                        error += (y[i] - y_pred[i]) ** 2
                error = error/n_samples
                return error

        # --------------- Parameters after estimation␣
 ↪---------------------------

        def parameters(self, fit_intercept = False):
                if(fit_intercept == True):
                        return self._w, self._b
                elif(fit_intercept == False):
                        return self._w
```

standard fit - using a plane to fit the lines - no feature transformations

[13]:
```
Linear_Regression = linear_model()
#z = np.square(df_train.x1)+ np.square(df_train.x2)
df_1 = pd.DataFrame({
        'x1' : df_train['x1'],
        'x2' : df_train['x2']
})

X1 = np.array(df_1[['x1','x2']])

m_1 = Linear_Regression.analytical_solution(X1, Y)
print('analytical_solution:\t', m_1)
pred_1 = Linear_Regression.predict(X1)
loss_11 = Linear_Regression.SSE(Y, pred_1)
print('SSE:\t',loss_11)
loss_12 = Linear_Regression.MSE(Y, pred_1)
print('MSE:\t',loss_12)

%matplotlib widget
figure = plt.figure(figsize=(5,5))
ax = plt.axes(projection = '3d')
ax.grid()
ax.plot3D(df_train.x1, df_train.x2, df_train.y, 'r.')
ax.plot3D(df_train.x1, df_train.x2, pred_1, 'b.')
plt.title('Standard Linear regression fit')
plt.legend(['Training Data', 'Standard Regression Fit'], loc = 'lower right')
ax.set_xlabel('x1')
```

```
ax.set_ylabel('x2')
ax.set_zlabel('y')
#ax.scatter(df_train.x1, df_train.x2, df_train.y, 'r.' )
plt.show()
```

```
analytical_solution:     [1.44599914 3.88421178]
SSE:      221020.65594915848
MSE:      221.02065594915848
```



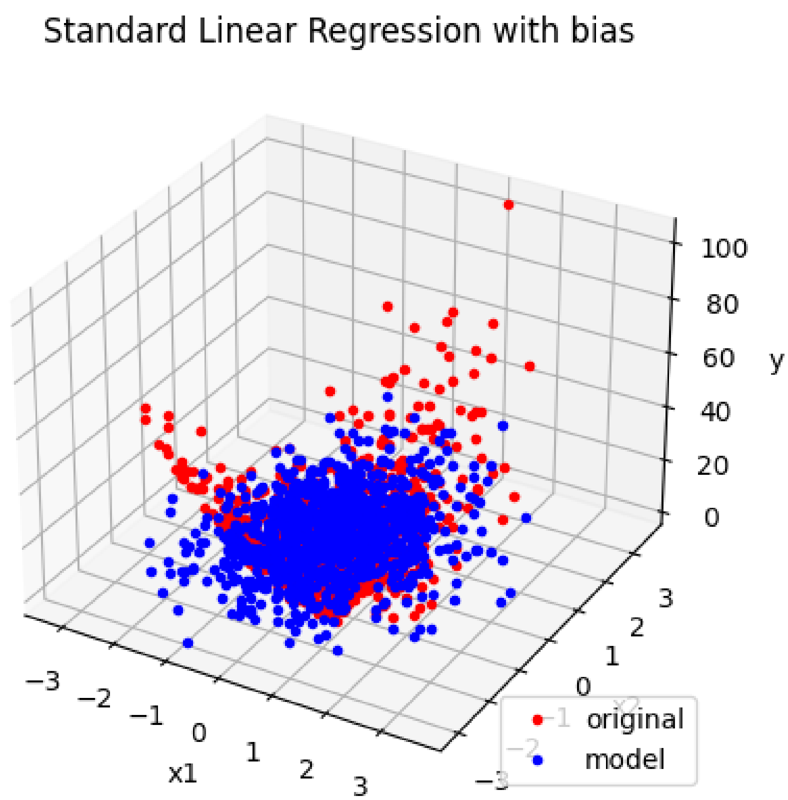
[14]:
```
plt.close()
```

standard fit - using a plane with intercept to fit the lines

[15]:
```
df_1 = pd.DataFrame({
        'x1' : df_train['x1'],
        'x2' : df_train['x2'],
        'bias': np.ones(1000)
})
```

```
X1 = np.array(df_1[['x1','x2','bias']])
print(X1.shape)

m_1 = Linear_Regression.analytical_solution(X1, Y)
print('analytical_solution:\t', m_1)
pred_1 = Linear_Regression.predict(X1)
loss_11 = Linear_Regression.SSE(Y, pred_1)
print('SSE:\t',loss_11)
loss_12 = Linear_Regression.MSE(Y, pred_1)
print('MSE:\t',loss_12)

%matplotlib widget
figure = plt.figure(figsize=(5,5))
ax = plt.axes(projection = '3d')
ax.grid()
ax.plot3D(df_train.x1, df_train.x2, df_train.y, 'r.')
ax.plot3D(df_train.x1, df_train.x2, pred_1, 'b.')
plt.title('Standard Linear Regression with bias')
plt.legend(['original', 'model'], loc = 'lower right')
#ax.scatter(df_train.x1, df_train.x2, df_train.y, 'r.' )
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')

plt.show()
```

```
(1000, 3)
analytical_solution:     [1.76570568 3.5215898  9.89400832]
SSE:     123364.85997994839
MSE:     123.36485997994839
```



[16]:
```
plt.close()
```

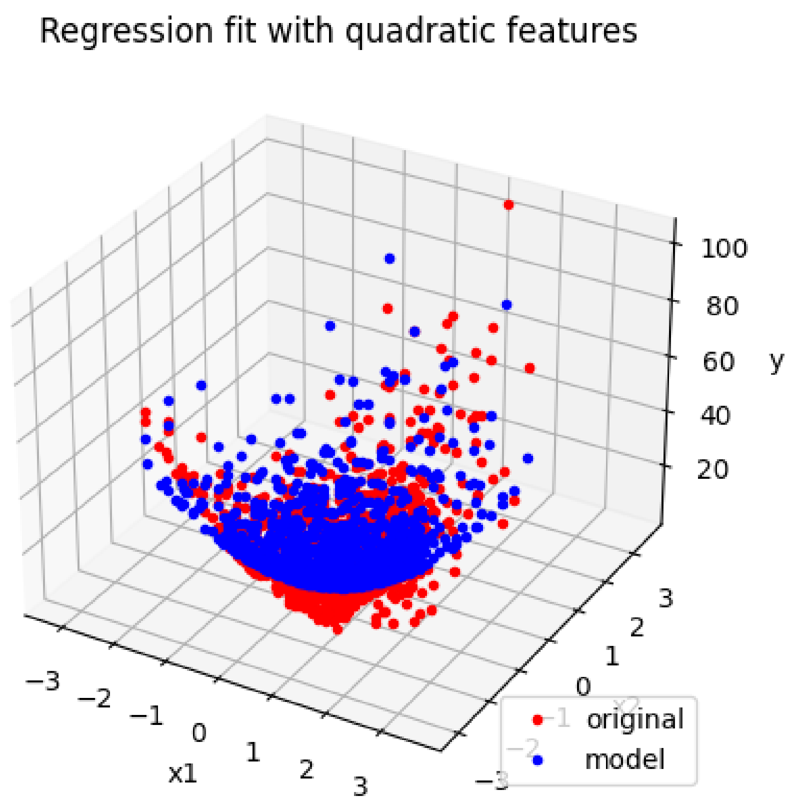
standard fit - using a plane to fit the lines - with quadratic feature transformations

From the figure, we can see that the ellipsoid is centered at origin thus (h,k) = (0, 0) from df.describe(), we get min and max values of (x, y, z). By subtituting, we get the values of lenght of major axis and minor axis as unknows, we get the values as: c = -81.3449371978518 , d = 90.0706857891388

[17]:
```
df_3 = pd.DataFrame({
        "x1" : np.square(df_train['x1'])*(1/90.0706),
        "x2" : np.square(df_train['x2'])*(-1* 1/81.3493)
})

X3 = np.array(df_3[["x1","x2"]])

m_3 = Linear_Regression.analytical_solution(X3, Y)
print('analytical_solution of weights:\t', m_3)
```

```python
pred_3 = Linear_Regression.predict(X3)
mbias_SSE = Linear_Regression.SSE(Y, pred_3)
print('SSE:\t', mbias_SSE)

mbias_SSE = Linear_Regression.MSE(Y, pred_3)
print('MSE:\t', mbias_SSE)

%matplotlib widget
figure = plt.figure(figsize=(5,5))
ax = plt.axes(projection = '3d')
ax.grid()
ax.plot3D(df_train.x1, df_train.x2, df_train.y, 'r.')
ax.plot3D(df_train.x1, df_train.x2, pred_3, 'b.')
plt.title('Regression fit with quadratic features')
plt.legend(['original', 'model'], loc = 'lower right')
#ax.scatter(df_train.x1, df_train.x2, df_train.y, 'r.' )
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
plt.show()
```

```
analytical_solution of weights:  [ 366.54469013 -442.44286378]
SSE:     45772.093411233334
MSE:     45.772093411233335
```



[18]:
```
plt.close()
```

From the above picture, we find that, the paraboloid is rotated to a certain angle , such that the fit and the data are not coinciding. the standard of a rotated ellipse is :

Reference: https://math.stackexchange.com/questions/426150/what-is-the-general-equation-of-the-ellipse-that-is-not-in-the-origin-and-rotate

using hunch, rotating by 30 degrees to 60 degrees in 1 degree at a time Using line search to do the rotation

[19]:
```
import math
from math import *
# for python, we need the angles to be in radians
theta_rad = [math.radians(i) for i in range(30, 61, 1)]

min_error = 1e+6
for i in theta_rad:
```

```
        df_transformed = pd.DataFrame({
                'x1' : (1/90.07) * (np.square((cos(i) * df_train['x1']) + ␣
↪(sin(i) * df_train['x2']))),
                'x2' : (-1 * 1/81.34) * (np.square((sin(i) * df_train['x1']) -␣
↪(cos(i) * df_train['x2']))),
                'bias': np.ones(1000)
        })

        X_transformed = np.array(df_transformed[['x1', 'x2', 'bias']])

        print('\nFor angle:\t',math.degrees(i))

        w = Linear_Regression.analytical_solution(X_transformed, Y)
        print('\nanalytical_solutionof weights:\t', w)

        predictions = Linear_Regression.predict(X_transformed)

        SSE = Linear_Regression.SSE(Y, predictions)
        print('SSE:\t', SSE)

        MSE = Linear_Regression.MSE(Y, predictions)
        print('MSE:\t', MSE)

        if(MSE < min_error):
                min_error = min(min_error, MSE)
print('\nMinimum Error:\t',min_error)
```

```
For angle:       29.999999999999996

analytical_solutionof weights:   [ 592.75855543 -206.26174905    0.9058721 ]
SSE:     30746.000995307182
MSE:     30.74600099530718

For angle:       31.0

analytical_solutionof weights:   [ 600.11152551 -200.00772053    0.90332342]
SSE:     29492.452551983763
MSE:     29.492452551983764

For angle:       32.0

analytical_solutionof weights:   [ 607.26444033 -193.93981014    0.90067568]
SSE:     28236.244001633677
MSE:     28.236244001633676

For angle:       33.0
```

```
analytical_solutionof weights:   [ 614.19797615 -188.07339333    0.89794209]
SSE:     26984.2229604796
MSE:     26.984222960479602

For angle:       34.0

analytical_solutionof weights:   [ 620.89265538 -182.42391243    0.89513673]
SSE:     25743.379037353658
MSE:     25.74337903735366

For angle:       35.0

analytical_solutionof weights:   [ 627.32900127 -177.00675112    0.89227448]
SSE:     24520.793745477436
MSE:     24.520793745477437

For angle:       36.0

analytical_solutionof weights:   [ 633.48770246 -171.83710155    0.8893709 ]
SSE:     23323.586092065016
MSE:     23.323586092065014

For angle:       37.0

analytical_solutionof weights:   [ 639.34978534 -166.92982595    0.88644214]
SSE:     22158.85445748084
MSE:     22.15885445748084

For angle:       38.0

analytical_solutionof weights:   [ 644.89679137 -162.29931487    0.88350482]
SSE:     21033.61551577352
MSE:     21.03361551577352

For angle:       39.0

analytical_solutionof weights:   [ 650.1109567  -157.95934456    0.88057592]
SSE:     19954.741077391354
MSE:     19.954741077391354

For angle:       40.0

analytical_solutionof weights:   [ 654.97539067 -153.92293581    0.87767263]
SSE:     18928.893846772466
MSE:     18.928893846772464

For angle:       41.0
```

```
analytical_solutionof weights:   [ 659.47425008 -150.2022173     0.87481217]
SSE:     17962.46317652637
MSE:     17.96246317652637

For angle:       42.0

analytical_solutionof weights:   [ 663.59290571 -146.80829578    0.8720117 ]
SSE:     17061.50196088861
MSE:     17.06150196088861

For angle:       43.0

analytical_solutionof weights:   [ 667.31809773 -143.75113608    0.86928812]
SSE:     16231.665839765488
MSE:     16.231665839765487

For angle:       44.0

analytical_solutionof weights:   [ 670.63807687 -141.03945333    0.86665791]
SSE:     15478.155877950965
MSE:     15.478155877950964

For angle:       45.0

analytical_solutionof weights:   [ 673.54272821 -138.68061971    0.864137  ]
SSE:     14805.665840480651
MSE:     14.805665840480652

For angle:       46.0

analytical_solutionof weights:   [ 676.0236751  -136.68058785    0.86174063]
SSE:     14218.335104734535
MSE:     14.218335104734535

For angle:       47.0

analytical_solutionof weights:   [ 678.0743608  -135.04383251    0.85948315]
SSE:     13719.708134744678
MSE:     13.719708134744678

For angle:       48.00000000000001

analytical_solutionof weights:   [ 679.69010646 -133.77331176    0.85737796]
SSE:     13312.701296843463
MSE:     13.312701296843462

For angle:       49.0
```

```
analytical_solutionof weights:   [ 680.86814394 -132.8704484     0.85543732]
SSE:     12999.577623521795
MSE:     12.999577623521795

For angle:       50.0

analytical_solutionof weights:   [ 681.60762324 -132.335132      0.8536723 ]
SSE:     12781.92994066611
MSE:     12.78192994066611

For angle:       51.0

analytical_solutionof weights:   [ 681.90959455 -132.16574129    0.85209267]
SSE:     12660.672569661414
MSE:     12.660672569661413

For angle:       52.0

analytical_solutionof weights:   [ 681.77696554 -132.35918613    0.85070682]
SSE:     12636.041608170679
MSE:     12.636041608170679

For angle:       53.0

analytical_solutionof weights:   [ 681.21443534 -132.91096798    0.84952173]
SSE:     12707.603589763099
MSE:     12.7076035897631

For angle:       54.0

analytical_solutionof weights:   [ 680.22840692 -133.81525728    0.8485429 ]
SSE:     12874.272130635134
MSE:     12.874272130635134

For angle:       55.0

analytical_solutionof weights:   [ 678.82688014 -135.06498579    0.84777437]
SSE:     13134.331998308364
MSE:     13.134331998308364

For angle:       56.0

analytical_solutionof weights:   [ 677.01932811 -136.65195172    0.84721871]
SSE:     13485.469888127425
MSE:     13.485469888127426

For angle:       57.00000000000001
```

```
analytical_solutionof weights:   [ 674.81655972 -138.56693526    0.84687704]
SSE:     13924.811072951294
MSE:     13.924811072951293

For angle:       58.00000000000001

analytical_solutionof weights:   [ 672.23057141 -140.79982201    0.84674906]
SSE:     14448.961002418975
MSE:     14.448961002418976

For angle:       59.00000000000001

analytical_solutionof weights:   [ 669.27439132 -143.33973178    0.8468331 ]
SSE:     15054.050871783453
MSE:     15.054050871783453

For angle:       59.99999999999999

analytical_solutionof weights:   [ 665.96191886 -146.17515019    0.8471262 ]
SSE:     15735.786156220904
MSE:     15.735786156220904

Minimum Error:   12.636041608170679
```

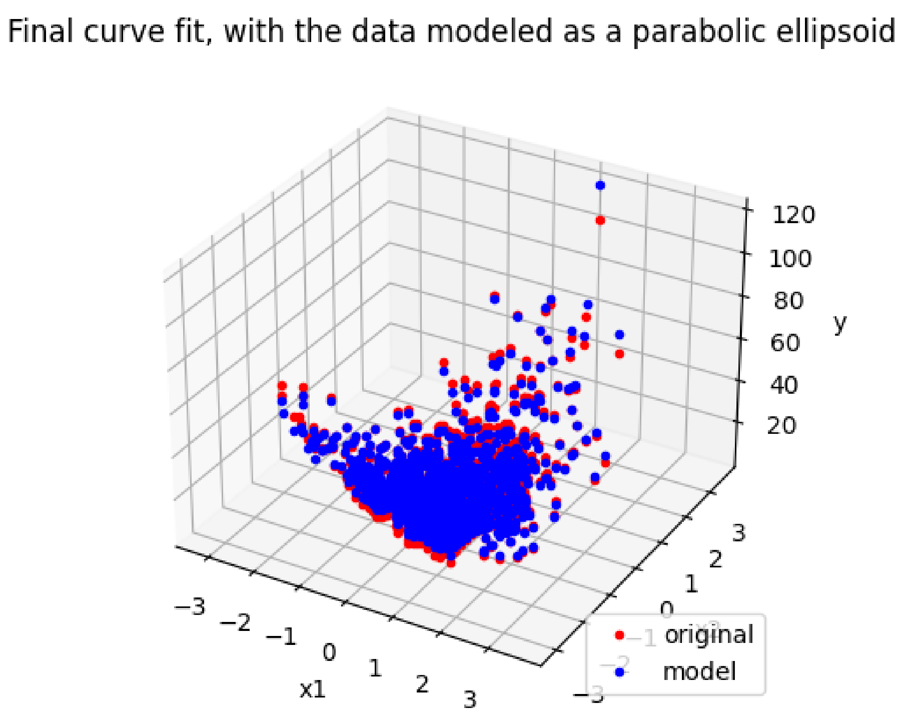
We see that for angle = 52 degrees, the error is minimum, but the plot is again tilted, thus extending the rotations to roll, pitch and yaw

[20]:
```
# Duplicate of the original training data
XY = np.array(df_train[['x1', 'x2', 'y']])

angle1 = 0.045 # alpha = +0.045 deg Roll
angle2 = -.5   # beta  = -0.5   deg Pitch
angle3 = -1.5  # gamma = -1.75  deg Yaw

x_rotation_matrix = [
        [ 1,                            0,                               0],
        [ 0, np.cos(math.radians(angle1)), -1*np.sin(math.radians(angle1))],
        [ 0, np.sin(math.radians(angle1)),    np.cos(math.radians(angle1))]
]

y_rotation_matrix = [
        [    np.cos(math.radians(angle2)), 0, np.sin(math.radians(angle2))],
        [                               0, 1,                            0],
        [ -1*np.sin(math.radians(angle2)), 0, np.cos(math.radians(angle2))]
]

z_rotation_matrix = [
```

```python
        [ np.cos(math.radians(angle3)), -1*np.sin(math.radians(angle3)), 0],
        [ np.sin(math.radians(angle3)),    np.cos(math.radians(angle3)), 0],
        [                            0,                               0, 1]
]

XY_transformed = XY @ y_rotation_matrix @ x_rotation_matrix @ z_rotation_matrix

df_new = pd.DataFrame(XY_transformed, columns = ['x1', 'x2', 'y'])
# df_new.head()

df_transformed = pd.DataFrame({
        'x1' : (1/90.07) * (np.square((cos(math.radians(52)) * df_new['x1']) + ␣
 ↪(sin(math.radians(52)) * df_new['x2']))),
        'x2' : (-1 * 1/81.34) * (np.square((sin(math.radians(52)) *␣
 ↪df_new['x1']) - (cos(math.radians(52)) * df_new['x2']))),
        'bias': np.ones(1000)
})

X_transformed = np.array(df_transformed[['x1', 'x2', 'bias']])

w = Linear_Regression.analytical_solution(X_transformed, Y)
print('\nanalytical_solutionof weights:\t', w)

predictions = Linear_Regression.predict(X_transformed)

SSE = Linear_Regression.SSE(Y, predictions)
print('SSE:\t', SSE)

MSE = Linear_Regression.MSE(Y, predictions)
print('MSE:\t', MSE)

%matplotlib widget
figure = plt.figure()
ax = plt.axes(projection = '3d')
ax.grid()
ax.plot3D(df_train.x1, df_train.x2, df_train.y, 'r.')
ax.plot3D(df_train.x1, df_train.x2, predictions, 'b.')
# ax.set_xlabel('x', labelpad=1)
# ax.set_ylabel('y', labelpad=1)
# ax.set_zlabel('z', labelpad=20)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')

plt.title('Final curve fit, with the data modeled as a parabolic ellipsoid')
plt.legend(['original', 'model'], loc = 'lower right')
#ax.scatter(df_train.x1, df_train.x2, df_train.y, 'r.' )
```

```
plt.show()
```

```
analytical_solutionof weights:   [ 637.61204116 -114.07473794    1.17226188]
SSE:      4778.569912372288
MSE:      4.778569912372288
```



[21]:
```
plt.close()
```

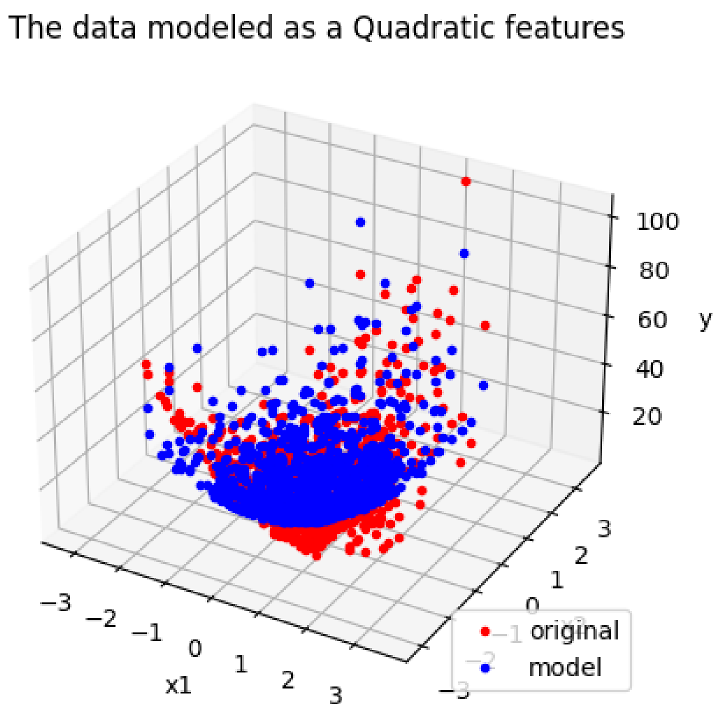
Voila!, it is a perfect fit! - clearly the features are quadratic and we can use a quadratic kernel (degree 2 version of polynomial kernel) or do quadratic feature transformations to get a similar fit!

Generalising the above fit to quadratic features

[24]:
```
df_transformed = pd.DataFrame({
         '1'                : np.ones(1000),
         'x1'     : sqrt(2)*df_train['x1'],
         'x2'     : sqrt(2)*df_train['x2'],
         '(x1)^2': np.square(df_train['x1']),
         '(x2)^2': np.square(df_train['x2']),
         'x1.x2' : sqrt(2)*np.vdot(df_train['x1'], df_train['x2'])
})
```

```python
X_transformed = np.array(df_transformed[['1', 'x1', 'x2', '(x1)^2', '(x2)^2',␣
 ↪'x1.x2']])

w = Linear_Regression.analytical_solution(X_transformed, Y)
print('\nanalytical_solution of weights:\t', w)

predictions = Linear_Regression.predict(X_transformed)

SSE = Linear_Regression.SSE(Y, predictions)
print('SSE:\t', SSE)

MSE = Linear_Regression.MSE(Y, predictions)
print('MSE:\t', MSE)

%matplotlib widget
figure = plt.figure()
ax = plt.axes(projection = '3d')
ax.grid()
ax.plot3D(df_train.x1, df_train.x2, df_train.y, 'r.')
ax.plot3D(df_train.x1, df_train.x2, predictions, 'b.')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')

plt.title('The data modeled as a Quadratic features')
plt.legend(['original', 'model'], loc = 'lower right')
#ax.scatter(df_train.x1, df_train.x2, df_train.y, 'r.' )
plt.show()
```

```
analytical_solution of weights:  [1.77965206 1.17645806 1.97202778 3.94698915
4.94687973 0.01655699]
SSE:     35341.919958649596
MSE:     35.3419199586496
```



[25]:
```
plt.close()
```

Thus, a quadratic kernel can be used to get better results, The generalised solution can be found in the main.py file!

[26]:
```
df_test = pd.read_csv('/Users/nandhakishorecs/Documents/IITM/Jul_Nov_2024/
↪DA5400_Fundamentals_of_Machine_Learning/Assignment1/Dataset/FMLA1Q1Data_test.
↪csv', header = None)
print('\nDescription of the training data:\n',df_test.describe())
df_test.columns = ['x1', 'x2', 'y']

print('\nColumns in training dataset:\t',df_test.columns)
Y = np.array(df_test.y)
print('\nLabel shape:\t',Y.shape)
```

```
Description of the training data:
                  0           1           2
count  100.000000  100.000000  100.000000
mean    -0.108502    0.037631    8.855249
std      1.088409    0.955979    9.098330
```

```
min     -2.771200   -2.132700    0.718040
25%     -0.819738   -0.642785    2.745725
50%     -0.036904   -0.011982    6.169500
75%      0.681887    0.674700   11.137250
max      3.012500    3.266200   54.237000

Columns in training dataset:     Index(['x1', 'x2', 'y'], dtype='object')

Label shape:     (100,)
```

[27]:
```python
# For cross validation quesiton
X_train = np.array(df_train[['x1', 'x2']])
X_test = np.array(df_test[['x1', 'x2']])
```

[28]:
```python
value1 = linalg.inv(dot(X_train, X_train.T))
value1
```

[28]:
```
array([[-3.32997753e+14, -6.73949052e+11,  5.80086435e+13, …,
         3.94687679e+14, -4.43335267e+14, -2.71505127e+14],
       [-4.41929109e+14,  1.85823473e+14,  4.88434874e+14, …,
         2.51859266e+14, -5.88950928e+13, -2.21969893e+14],
       [-1.12105726e+14, -5.60301374e+13, -2.69181396e+14, …,
        -8.54334025e+14,  5.03351603e+14,  1.21934371e+14],
       …,
       [-5.07149428e+12, -6.24038596e+13, -1.36570765e+14, …,
        -1.36852518e+14, -4.15881273e+13,  1.51411957e+13],
       [ 2.01974325e+14, -7.52082984e+13, -1.49076793e+14, …,
         8.76159602e+12,  5.86294173e+13,  9.51481493e+13],
       [ 8.40764163e+13, -4.83646744e+13, -1.74706164e+14, …,
        -7.33627821e+13, -4.16879311e+12,  7.87862203e+13]])
```

[29]:
```python
value2 = linalg.inv(dot(X_test, X_test.T))
value2
```

[29]:
```
array([[ 3.27796607e+15,  1.22114204e+16, -8.52770097e+14, …,
         5.97710932e+15,  8.13151873e+15, -5.11940705e+15],
       [-1.77345280e+15, -3.91195804e+14, -1.62872475e+15, …,
        -2.20518653e+15, -3.92350967e+15,  2.72716447e+15],
       [-8.07090654e+13, -1.41652632e+14, -6.47740302e+14, …,
        -1.82656427e+15,  1.10102148e+15,  1.07336937e+14],
       …,
       [-6.99984117e+15, -1.16200828e+16,  2.21329395e+15, …,
        -5.73454253e+15, -9.60113319e+15,  5.85060834e+15],
       [ 1.73924086e+15,  3.14661930e+15, -7.03649543e+13, …,
         2.67536010e+15,  4.15744411e+15, -3.29887042e+15],
       [ 3.49291343e+14,  2.55210481e+14,  3.73146083e+13, …,
        -2.25699355e+14, -1.76575232e+14,  2.47088623e+14]])
```

the variance of test and train data are very similar !

```
[ ]:
```