CarPricePrediction

January 8, 2024

Problem Statement

- Given the attributes of the customers, how much is the customer willing to pay ### Intuition
- Regression problem ### Implementation
- Using ANN(tensorflow)

[4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
import tensorflow.keras # keras is an API on top of tensorflow
from keras.models import Sequential # Building the model sequential..
from keras.layers import Dense # Since it is backpropagated..
```

[5]:
```
# importing data set
df = pd.read_csv('Car_Purchasing_Data.csv',encoding= 'ISO-8859-1')
df.shape
```

[5]:
```
(500, 9)
```

[6]:
```
df.head(5)
```

[6]:
```
       Customer Name                                    Customer e-mail  \
0      Martina Avila  cubilia.Curae.Phasellus@quisaccumsanconvallis.edu
1      Harlan Barnes                                eu.dolor@diam.co.uk
2    Naomi Rodriquez  vulputate.mauris.sagittis@ametconsectetueradip…
3    Jade Cunningham                            malesuada@dignissim.com
4       Cedric Leach     felis.ullamcorper.viverra@egetmollislectus.net

         Country  Gender        Age  Annual Salary  Credit Card Debt  \
0       Bulgaria       0  41.851720    62812.09301      11609.380910
1         Belize       0  40.870623    66646.89292       9572.957136
2        Algeria       1  43.152897    53798.55112      11160.355060
3   Cook Islands       1  58.271369    79370.03798      14426.164850
4         Brazil       1  57.313749    59729.15130       5358.712177
```

```
     Net Worth  Car Purchase Amount
0  238961.2505          35321.45877
1  530973.9078          45115.52566
2  638467.1773          42925.70921
3  548599.0524          67422.36313
4  560304.0671          55915.46248
```

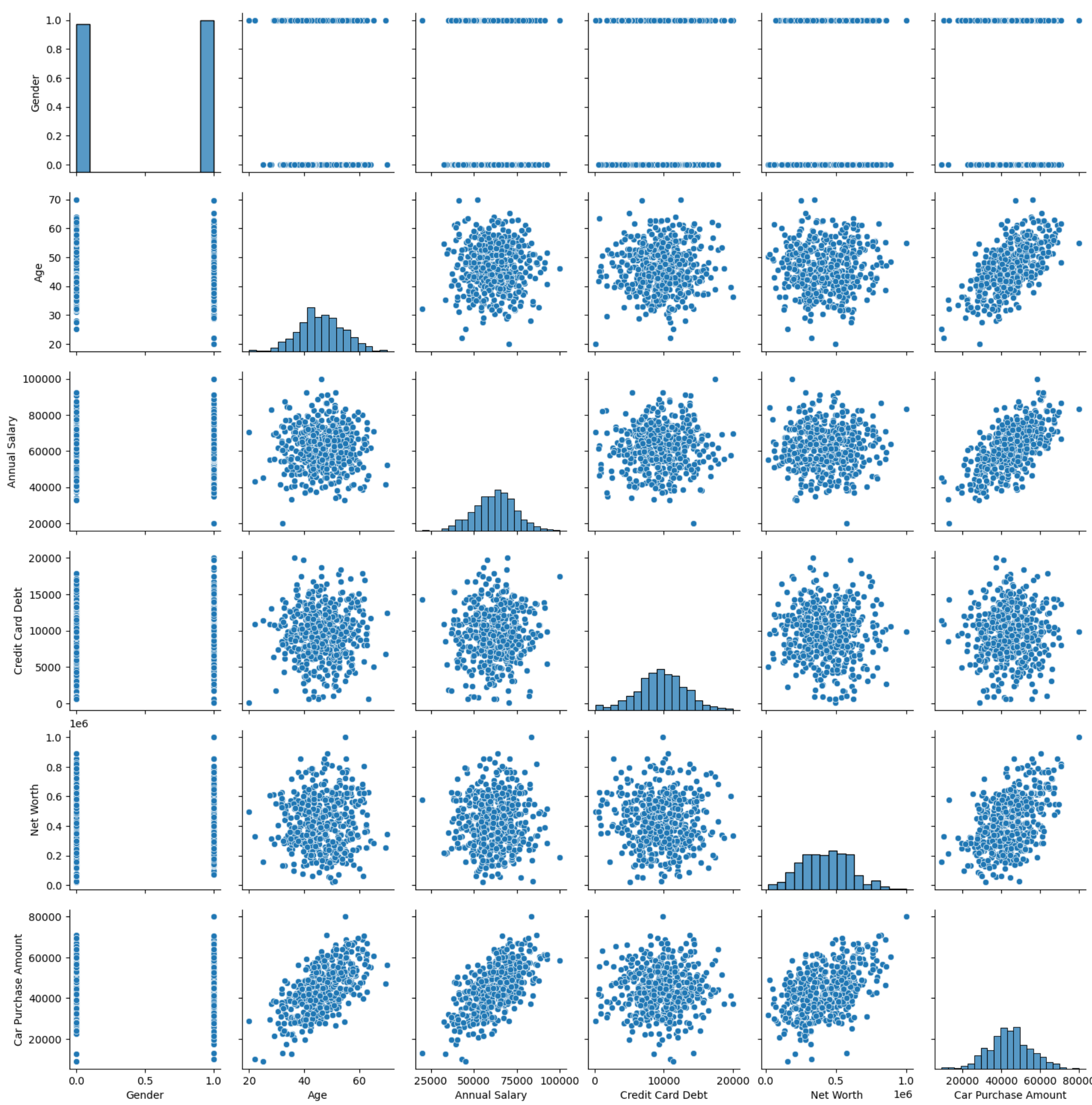
[7]:
```
df.columns
```

[7]:
```
Index(['Customer Name', 'Customer e-mail', 'Country', 'Gender', 'Age',
       'Annual Salary', 'Credit Card Debt', 'Net Worth',
       'Car Purchase Amount'],
      dtype='object')
```

Visualization

[8]:
```
sns.pairplot(df)
```

[8]:
```
<seaborn.axisgrid.PairGrid at 0x16ff6c032d0>
```



0.0.1 Data Cleaning

- Dropping customer name, email and country that doesn't have any effect on the data
- X-> denotes the input features, so drop Car purchase amount as well

Separating Features and Target

[9]:
```
X = df.drop(['Customer Name','Customer e-mail','Country','Car Purchase␣
↪Amount'],axis=1)
X.head(5)
```

[9]:
```
   Gender        Age  Annual Salary  Credit Card Debt    Net Worth
0       0  41.851720    62812.09301      11609.380910  238961.2505
1       0  40.870623    66646.89292       9572.957136  530973.9078
```

```
2       1  43.152897    53798.55112      11160.355060  638467.1773
3       1  58.271369    79370.03798      14426.164850  548599.0524
4       1  57.313749    59729.15130       5358.712177  560304.0671
```

[10]:
```python
Y = df['Car Purchase Amount']
Y.head(5)
```

[10]:
```
0    35321.45877
1    45115.52566
2    42925.70921
3    67422.36313
4    55915.46248
Name: Car Purchase Amount, dtype: float64
```

[11]:
```python
# Shape of feature and target....
X.shape,Y.shape
```

[11]:
```
((500, 5), (500,))
```

Perform Normalization

- Since ANN works better after on normalized data
- Converts values from 0 to 1

- Scaling features

[12]:
```python
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

[13]:
```python
X_scaled[:5]
```

[13]:
```
array([[0.        , 0.4370344 , 0.53515116, 0.57836085, 0.22342985],
       [0.        , 0.41741247, 0.58308616, 0.476028  , 0.52140195],
       [1.        , 0.46305795, 0.42248189, 0.55579674, 0.63108896],
       [1.        , 0.76542739, 0.74212547, 0.71990778, 0.53938679],
       [1.        , 0.74627499, 0.49661439, 0.26425689, 0.55133068]])
```

[14]:
```python
# Returns the maximum and minimum value of each feature
scaler.data_max_,scaler.data_min_
```

[14]:
```
(array([1.e+00, 7.e+01, 1.e+05, 2.e+04, 1.e+06]),
 array([    0.,    20., 20000.,   100., 20000.]))
```

- Reshaping target since it is an 1D array

[15]:
```python
Y = Y.values.reshape(-1,1)
```

[16]:
```python
# If the feature is an 1d array it should be reshaped such that it has n rows␣
↪and 1 columns..
```

```
# to train tensorflow model
Y.shape
```

[16]: (500, 1)

- Scaling Target

[17]:
```
# Scaeling the target
Y_scaled = scaler.fit_transform(Y)
Y_scaled[:5]
```

[17]:
```
array([[0.37072477],
       [0.50866938],
       [0.47782689],
       [0.82285018],
       [0.66078116]])
```

Training Model

- Splitting data

[18]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled,Y_scaled)
```

[19]:
```
X_train.shape,X_test.shape
```

[19]: ((375, 5), (125, 5))

- Building Model

[20]:
```
# 25 -> neurons in the hidden layer...
# input_dim -> No. of features..
# activation -> activation function...
model = Sequential([
    #Hidden layer - 1
    #input_dim is optional..
    Dense(25, input_dim = 5,activation = 'relu'),
    # Another hidden layer..
    Dense(25, activation = 'relu'),
    # Output layer...
    # 'linear' - for regression
    Dense(1, activation = 'linear')
])
```

```
WARNING:tensorflow:From C:\Users\samli\AppData\Roaming\Python\Python311\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.
```

[21]:
```
model.summary()
#150 = 5 x 25 + 25
#650 = 25 x 25 + 25
#26  = 25 x 1 + 1
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 25)                150

 dense_1 (Dense)             (None, 25)                650

 dense_2 (Dense)             (None, 1)                 26

=================================================================
Total params: 826 (3.23 KB)
Trainable params: 826 (3.23 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[22]:
```
model.compile(optimizer= 'adam', loss= 'mean_squared_error')
```

```
WARNING:tensorflow:From C:\Users\samli\AppData\Roaming\Python\Python311\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

[24]:
```
epochs_hist = model.fit(X_train,Y_train, epochs = 20, batch_size = 25, verbose␣
 ↪= 1, validation_split = 0.2)
# validation_split -> To avoid overfitting
```

```
Epoch 1/20
12/12 [==============================] - 0s 17ms/step - loss: 0.0016 - val_loss:
0.0016
Epoch 2/20
12/12 [==============================] - 0s 9ms/step - loss: 0.0014 - val_loss:
0.0014
Epoch 3/20
12/12 [==============================] - 0s 9ms/step - loss: 0.0013 - val_loss:
0.0013
Epoch 4/20
12/12 [==============================] - 0s 10ms/step - loss: 0.0011 - val_loss:
0.0011
Epoch 5/20
12/12 [==============================] - 0s 7ms/step - loss: 9.6032e-04 -
val_loss: 0.0010
Epoch 6/20
12/12 [==============================] - 0s 9ms/step - loss: 8.4678e-04 -
```

```
val_loss: 9.1730e-04
Epoch 7/20
12/12 [==============================] - 0s 9ms/step - loss: 7.7136e-04 -
val_loss: 8.4126e-04
Epoch 8/20
12/12 [==============================] - 0s 8ms/step - loss: 6.4754e-04 -
val_loss: 7.3718e-04
Epoch 9/20
12/12 [==============================] - 0s 8ms/step - loss: 5.7045e-04 -
val_loss: 6.7446e-04
Epoch 10/20
12/12 [==============================] - 0s 8ms/step - loss: 5.0783e-04 -
val_loss: 6.1279e-04
Epoch 11/20
12/12 [==============================] - 0s 9ms/step - loss: 4.4649e-04 -
val_loss: 5.5932e-04
Epoch 12/20
12/12 [==============================] - 0s 10ms/step - loss: 4.0033e-04 -
val_loss: 5.0437e-04
Epoch 13/20
12/12 [==============================] - 0s 11ms/step - loss: 3.6614e-04 -
val_loss: 4.6812e-04
Epoch 14/20
12/12 [==============================] - 0s 9ms/step - loss: 3.3139e-04 -
val_loss: 4.2996e-04
Epoch 15/20
12/12 [==============================] - 0s 9ms/step - loss: 3.1255e-04 -
val_loss: 4.0578e-04
Epoch 16/20
12/12 [==============================] - 0s 8ms/step - loss: 2.8246e-04 -
val_loss: 3.7452e-04
Epoch 17/20
12/12 [==============================] - 0s 8ms/step - loss: 2.5609e-04 -
val_loss: 3.4812e-04
Epoch 18/20
12/12 [==============================] - 0s 8ms/step - loss: 2.3939e-04 -
val_loss: 3.3054e-04
Epoch 19/20
12/12 [==============================] - 0s 7ms/step - loss: 2.2756e-04 -
val_loss: 3.1276e-04
Epoch 20/20
12/12 [==============================] - 0s 7ms/step - loss: 2.1449e-04 -
val_loss: 2.9999e-04
```

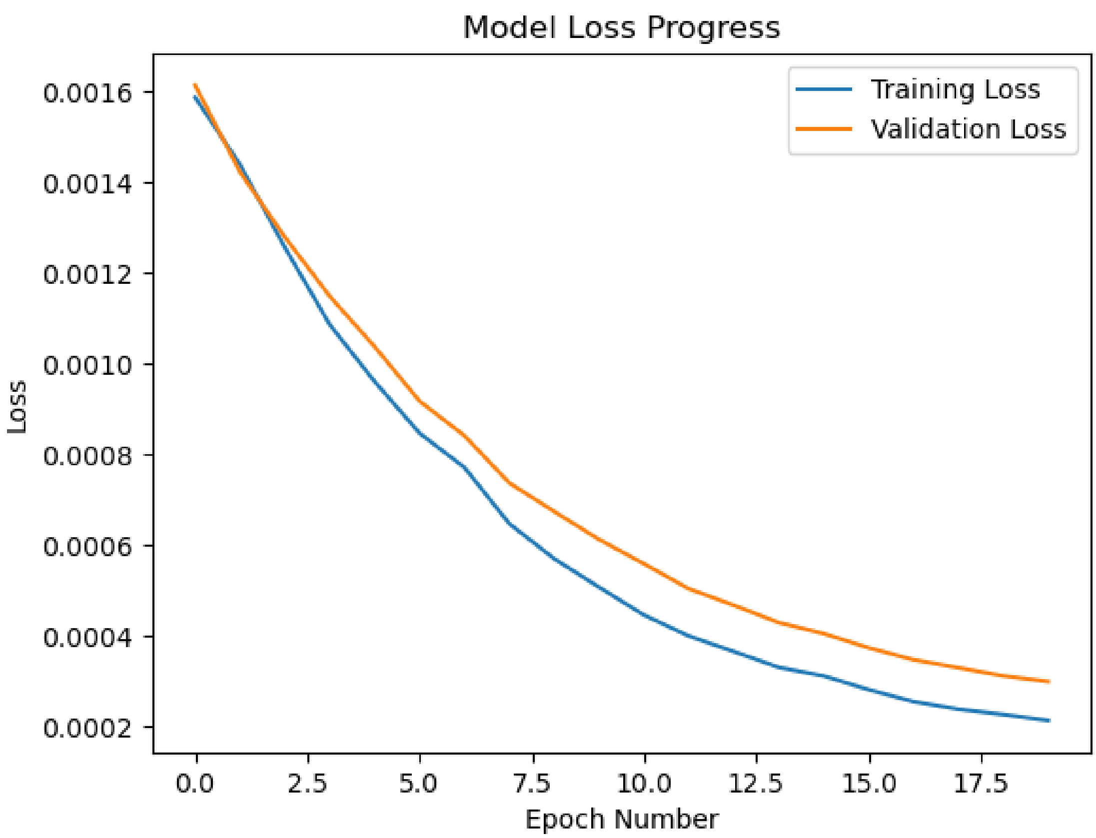
Model Evaluation

[25]:
```
epochs_hist.history.keys()
```

[25]: dict_keys(['loss', 'val_loss'])

[27]:
```
plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])
plt.title('Model Loss Progress')
plt.ylabel('Loss')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss','Validation Loss'])
```

[27]: <matplotlib.legend.Legend at 0x16ff992a950>



Prediction

- Considering Random data point

[28]:
```
#Gender, Age, Annual Salary, Credit Card Debt, Net worth
X_test = np.array([[1, 50, 5000, 10000, 600000]])
y_prediction = model.predict(X_test)
print(f'Expected purchase amount: {y_prediction}')
```

```
1/1 [==============================] - 0s 221ms/step
```

```
Expected purchase amount: [[179020.28]]
```