SVM - Regression

SHUMBUL ARIFA \ 181CO152

Task

Performing SVM on a regression dataset.

1. Linear Kernel
2. Polynomial Kernel
3. Radial Basis Function (RBF) kernel

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as plt
```

In [2]:
```
df = pd.read_csv("Movie_regression.csv")
df.head()
```

Out[2]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Critic_
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Marketing expense    506 non-null    float64
 1   Production expense   506 non-null    float64
 2   Multiplex coverage   506 non-null    float64
 3   Budget               506 non-null    float64
 4   Movie_length         506 non-null    float64
 5   Lead_ Actor_Rating   506 non-null    float64
 6   Lead_Actress_rating  506 non-null    float64
 7   Director_rating      506 non-null    float64
 8   Producer_rating      506 non-null    float64
 9   Critic_rating        506 non-null    float64
 10  Trailer_views        506 non-null    int64  
 11  3D_available         506 non-null    object 
 12  Time_taken           494 non-null    float64
 13  Twitter_hastags      506 non-null    float64
 14  Genre                506 non-null    object 
 15  Avg_age_actors       506 non-null    int64  
 16  Num_multiplex        506 non-null    int64  
 17  Collection           506 non-null    int64  
dtypes: float64(12), int64(4), object(2)
memory usage: 71.3+ KB
```

Data Cleaning and preprocessing

1. time_taken has some missing values
2. 3D_available and Genre -> object type (string)

In [4]:
```
mean = df['Time_taken'].mean()
mean
```

Out[4]: 157.39149797570855

In [5]:
```
df['Time_taken'].fillna(value = mean, inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Marketing expense    506 non-null    float64
 1   Production expense   506 non-null    float64
 2   Multiplex coverage   506 non-null    float64
 3   Budget               506 non-null    float64
 4   Movie_length         506 non-null    float64
 5   Lead_ Actor_Rating   506 non-null    float64
 6   Lead_Actress_rating  506 non-null    float64
 7   Director_rating      506 non-null    float64
 8   Producer_rating      506 non-null    float64
 9   Critic_rating        506 non-null    float64
 10  Trailer_views        506 non-null    int64  
 11  3D_available         506 non-null    object 
 12  Time_taken           506 non-null    float64
 13  Twitter_hastags      506 non-null    float64
 14  Genre                506 non-null    object 
 15  Avg_age_actors       506 non-null    int64  
 16  Num_multiplex        506 non-null    int64  
 17  Collection           506 non-null    int64  
dtypes: float64(12), int64(4), object(2)
memory usage: 71.3+ KB
```

In [6]:
```
## 3D-available and Genre
# ### Using dummy variable creation

# df = pd.get_dummies(df, columns = ["3D_available", "Genre"])
# df.info()
obj_df = df.select_dtypes(include=['object']).copy()
obj_df.head()
```

Out[6]:

	3D_available	Genre
0	YES	Thriller
1	NO	Drama
2	NO	Comedy
3	YES	Drama
4	NO	Drama

In [7]:
```
## if any null value is present in those rows
obj_df[obj_df.isnull().any(axis=1)]
```

Out[7]:

3D_available	Genre

In [8]:
```
# ## if it was present in column "c"
# obj_df["c"].value_counts()
# obj_df = obj_df.fillna({"c": "NEW_NAME"})
```

In [9]:
```
## replace
cleanup_nums = {"3D_available":     {"YES": 1, "NO": 0},
                "Genre": {"Thriller": 0, "Drama": 1, "Comedy": 2, "Action": 3}}
```

In [10]:
```
## replace only once!
df = df.replace(cleanup_nums)
df.info()

## done
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Marketing expense    506 non-null    float64
 1   Production expense   506 non-null    float64
 2   Multiplex coverage   506 non-null    float64
 3   Budget               506 non-null    float64
 4   Movie_length         506 non-null    float64
 5   Lead_ Actor_Rating   506 non-null    float64
 6   Lead_Actress_rating  506 non-null    float64
 7   Director_rating      506 non-null    float64
 8   Producer_rating      506 non-null    float64
 9   Critic_rating        506 non-null    float64
 10  Trailer_views        506 non-null    int64  
 11  3D_available         506 non-null    int64  
 12  Time_taken           506 non-null    float64
 13  Twitter_hastags      506 non-null    float64
 14  Genre                506 non-null    int64  
 15  Avg_age_actors       506 non-null    int64  
 16  Num_multiplex        506 non-null    int64  
 17  Collection           506 non-null    int64  
dtypes: float64(12), int64(6)
memory usage: 71.3 KB
```

X_y split

In [11]:
```
X = df.loc[:,df.columns!="Collection"]
# All cols except collection

type(X)
```

Out[11]: pandas.core.frame.DataFrame

In [12]:
```
X.head()
```

Out[12]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Critic_
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [13]:
```
X.shape
```

Out[13]: (506, 17)

In [14]:
```
y = df["Collection"]
type(y)
```

Out[14]: pandas.core.series.Series

In [15]:
```
y.head()
```

Out[15]:
```
0    48000
1    43200
2    69400
3    66800
4    72400
Name: Collection, dtype: int64
```

In [16]:
```
y.shape
```

Out[16]: (506,)

Test-Train Split

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

In [19]:
```
X_train.head()
```

Out[19]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Criti
220	27.1618	67.40	0.493	38612.805	162.0	8.485	8.640	8.485	8.670	
71	23.1752	76.62	0.587	33113.355	91.0	7.280	7.400	7.290	7.455	
240	22.2658	64.86	0.572	38312.835	127.8	6.755	6.935	6.800	6.840	
6	21.7658	70.74	0.476	33396.660	140.1	7.065	7.265	7.150	7.400	
417	538.8120	91.20	0.321	29463.720	162.6	9.135	9.305	9.095	9.165	

In [20]:
```
y_test.head()
```

Out[20]:
```
329     45200
371    100000
219     46000
403     16600
78      42400
Name: Collection, dtype: int64
```

In [21]:
```
X_train.shape
```

Out[21]: (404, 17)

Standardizing Data

- Coverting mean and variance close to 0 and 1, for each variable.
- SVM only gives correct result when we standardize our data!
- Ways: StandardScaler, MinMax scaler

In [22]:
```
from sklearn.preprocessing import StandardScaler
```

In [23]:
```
sc = StandardScaler().fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

In [24]:
```
X_test_std

# here, we only need to std our X data, not y
```

Out[24]:
```
array([[-0.40835869, -1.12872913,  0.83336883, ...,  0.71069324,
         1.12308956, -0.88738582],
       [ 0.71925111,  0.9988844 , -0.65283979, ...,  0.71069324,
        -1.15123717,  0.60896159],
       [-0.40257488,  0.39610829,  0.05115377, ...,  0.71069324,
        -1.47614099,  0.15147958],
       ...,
       [-0.3982601 , -0.85812418,  0.89420778, ..., -1.12982003,
        -0.7451074 , -1.01128719],
       [-0.39934279, -0.07637654,  0.58132175, ...,  0.71069324,
        -2.93820817, -0.99222544],
       [-0.40088071, -0.36702631,  0.31189212, ...,  1.63094988,
         0.71695979, -0.41084206]])
```

All decimals, scales of values changed -> uniform scale

Now, we can perform SVM

Performing SVM Classification

Linear Kernel

In [146]:
```
from sklearn.svm import SVR
svr = SVR(kernel='linear', C=500)
```

In [147]:
```
svr.fit(X_train_std, y_train)
```

Out[147]:
```
SVR(C=500, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Predict values using trained model

In [148]:
```
y_test_pred = svr.predict(X_test_std)
y_train_pred = svr.predict(X_train_std)
```

In [149]:
```
y_test_pred
```

Out[149]:
```
array([53748.13451391, 42540.50285611, 47517.42636814, 18927.45325836,
       47310.50867217, 40486.82922495, 34098.84435909, 43309.94120255,
       31078.23199081, 47838.16708872, 13361.57747337, 38288.1606548 ,
       37479.04189229,  9347.34909182, 65669.47486051, 60399.34345442,
       39061.81719881, 65412.25720215, 56810.9743022 , 43739.00070916,
       54114.23351522, 39978.70171033, 42383.96580219, 58178.80392456,
       43341.71665562, 14140.7167983 , 41512.60873727, 29820.28316867,
       74632.28035281, 45350.96054436, 35974.47356514, 36543.56496454,
       38640.74878297, 41392.68987408, 52489.07567473, 36111.47392681,
       24287.35356429, 36927.40176622, 37114.08799632, 34794.69877637,
       47092.46041122, 44971.31893491, 45078.62157286, 28705.10232012,
       50884.96672398, 46566.41946939, 33136.38621284, 40390.48867361,
       16019.05224311, 51031.10336878, 41339.67397913, 39122.55556566,
       48049.18605239, 66638.18579722, 26159.36247314, 41651.13567981,
       40438.14471749, 35005.15327306, 22605.55838049, 39737.96255962,
       41739.75840988, 41944.43712446, 64849.28603378, 64406.32941646,
       28467.85387942, 63833.57920213, 36672.90921912, 38563.10540505,
       41162.47958782, 47146.37838462, 46804.01253059, 48690.06089547,
       56849.69858922, 59986.67142211, 48991.21972999, 11324.11523815,
       73117.11823948, 44759.02314042, 58711.14585716, 37353.55688314,
       50961.97958875, 43065.12392831, 32948.88694092, 80052.15115569,
       78961.48366306, 48004.40193968, 51609.70797375, 30091.6351161 ,
       48321.42688865, 38160.89549256, 34118.65986045, 34364.82405189,
       43924.77224906, 59441.27770036, 44529.28679885, 42131.41673494,
       -4736.54768801, 51784.33327662, 36611.55551474, 33185.59458753,
       49229.89669031, 49470.78584054])
```

Model Performance

In [150]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

In [151]:
```
mean_squared_error(y_test, y_test_pred)
```

Out[151]: 159739968.21606734

In [152]:
```
r2_score(y_train, y_train_pred)
```

Out[152]: 0.7141279694523317

In [153]:
```
r2_score(y_test, y_test_pred)
```

Out[153]: 0.5037970011487986

Polynomial kernel

In [138]:
```
svr = SVR(kernel='poly', C=100000)
```

In [139]:
```
svr.fit(X_train_std, y_train)
```

Out[139]:
```
SVR(C=100000, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='poly', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Predict values using trained model

In [140]:
```
y_test_pred = svr.predict(X_test_std)
y_train_pred = svr.predict(X_train_std)
```

In [141]:
```
y_test_pred
```

Out[141]:
```
array([ 53945.06470276,  40019.81186347,  45013.44020017,  16473.70563161,
        39034.26966912,  38405.81518086,  37551.37327445,  41583.26693188,
        40501.41101511,  39959.03073358,   7260.23644352,  34585.91312818,
        33704.58535356,   8111.82313632,  82544.95396017,  61657.98789323,
        41003.66568077,  70133.16359053,  50402.25934686,  44768.14339885,
        46138.15073489,  46016.67195208,  34400.37018254,  54175.90317684,
        37924.5793689 ,  40812.65537599,  39827.89558469,  46199.74803236,
        91181.61210004,  42433.7899923 ,  33459.55407488,  32507.54901555,
        43665.2842708 ,  49988.36873724,  48501.82532205,  38609.21263118,
         9311.61749637,  41945.51915059,  37853.89130937,  34919.43194728,
        43858.38840063,  41931.41233793,  39439.53643861,  29012.86103898,
        45289.08641938,  45320.80727479,  40508.58725213,  42835.77644422,
         7421.65346006,  48189.47489161,  48977.81426194,  38000.917722  ,
        51520.94682695, 119150.1040879 ,  15944.29486146,  31244.3269301 ,
        24608.73593043,  34086.67375656,  64066.80845315,  40477.05095669,
        19639.10982262,  41389.04896907,  59744.69162649,  61771.25770938,
        37097.22697959,  60970.28863153,  37041.80751019,  38224.41846005,
        45078.59714338,  49169.83130621,  46792.92134314,  50400.96267509,
        52368.86832454,  60074.75993461,  44900.40418321,  11039.23010175,
        77040.54018372,  43928.39023456,  60398.35058773,  35286.19780383,
        56860.11073131,  47852.58644365,  25526.16173764,  87261.85938472,
       102662.24008638,  46218.89853112,  53894.25759941,  34473.39669641,
        41271.58551262,  28636.2850954 ,  46040.2897106 ,  21960.22983974,
        39484.28701741,  56994.14198523,  37594.64540993,  38913.06228559,
        12938.43606188,  47299.85471528,  33753.68696983,  25120.41106915,
        47901.97419923,  43717.68234851])
```

Model Performance

In [142]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

In [143]:
```
mean_squared_error(y_test, y_test_pred)
```

Out[143]: 160821631.58881903

In [144]:
```
r2_score(y_train, y_train_pred)
```

Out[144]: 0.9175322751758398

In [145]:
```
r2_score(y_test, y_test_pred)
```

Out[145]: 0.5004370116902999

RBF Kernel

In [195]:
```
svr = SVR(kernel='rbf', gamma = 0.05, C=100000)
```

In [196]:
```
svr.fit(X_train_std, y_train)
```

Out[196]:
```
SVR(C=100000, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.05,
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Predict values using trained model

In [197]:
```
y_test_pred = svr.predict(X_test_std)
y_train_pred = svr.predict(X_train_std)
```

In [198]:
```
y_test_pred
```

Out[198]:
```
array([53406.80210336, 46899.22039025, 48870.82315743, 18535.04836896,
       44883.57709382, 32604.54997274, 40001.61745006, 41720.64234004,
       51185.9575827 , 37749.33239083, 20252.48753031, 28790.04198996,
       26825.67725239, 16571.19245305, 87283.81235302, 66911.1640002 ,
       46628.56911194, 66823.29500983, 56282.77886105, 43999.61822059,
       48569.37819502, 45346.06904285, 38965.11965307, 55497.92624713,
       40023.38548795, 58672.26084795, 32726.78716029, 34405.80929691,
       79053.95275631, 45344.28444738, 30397.44314203, 35233.95160411,
       39879.90775495, 40035.22274539, 51568.40551971, 41079.76277026,
       14598.35871337, 51024.59037454, 34359.87096268, 27715.62673045,
       45858.77374849, 37954.62067894, 42532.46713566, 38590.11681156,
       49306.35044609, 41187.66408613, 43542.59408608, 34182.56800787,
       26238.19435436, 53032.38627988, 34801.69600807, 40302.7722779 ,
       46361.92443425, 72762.87598388, 21361.90190354, 46649.92485622,
       39559.92779659, 37808.61723158, 32173.54555302, 38138.1296051 ,
       41330.52240551, 41595.84067014, 68282.77170597, 59567.90381865,
       43928.25687   , 62359.90956385, 27334.35687868, 47285.73745374,
       36147.00159879, 43761.69484576, 47085.90984631, 47919.57491612,
       49572.61092247, 58851.97253468, 45526.263979  , 28233.56192847,
       96251.71395952, 44047.13984058, 60106.48704918, 38842.13979557,
       50441.20547465, 40065.82943553, 32448.69593166, 99515.87688829,
       85683.47620792, 46398.22586856, 45297.0005115 , 32922.45766371,
       44518.03689238, 28869.66173132, 43243.81741431, 18083.80574704,
       40196.01202102, 59741.12767828, 45972.89477721, 39083.24671591,
       18279.78949403, 44971.5276282 , 28705.39459964, 33589.57330489,
       60606.71430786, 42694.60754686])
```

Model Performance

In [199]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

In [200]:
```
mean_squared_error(y_test, y_test_pred)
```

Out[200]: 103113366.61472107

In [201]:
```
r2_score(y_train, y_train_pred)
```

Out[201]: 0.9603484140717995

In [202]:
```
r2_score(y_test, y_test_pred)
```

Out[202]: 0.6796971834459051