Support Vector Machine

Using Gridsearch CV to improve Accuracy of the model.

SVR

Pipeline of ML

1) Data ingesion

2) EDA

3) Preprocessing

4) model (mathematical equation) => SVM (1D to 2D) => SVC or SVR

5) Evaluation (performance metrics or confuse metrics or AUC / ROC)

____SVC => For Classification problems

____SVR => For Regression problems

In [21]:
```
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
```

In [74]:
```
import pandas as pd
data = pd.read_csv("https://raw.githubusercontent.com/srinivasav22/Graduate-Admis
data.head()
```

Out[74]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

In [75]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          500 non-null    int64  
 2   TOEFL Score        500 non-null    int64  
 3   University Rating  500 non-null    int64  
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

In [76]:
```
# All the columns are in integer or float type and having no null values
```

In [77]:
```
# lets drop the serial no column which is not needed in the dataframe
data.drop(['Serial No.'], axis=1, inplace=True)
data.head()
```

Out[77]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [78]:
```
data.describe().T
```

Out[78]:

	count	mean	std	min	25%	50%	75%	max
GRE Score	500.0	316.47200	11.295148	290.00	308.0000	317.00	325.00	340.00
TOEFL Score	500.0	107.19200	6.081868	92.00	103.0000	107.00	112.00	120.00
University Rating	500.0	3.11400	1.143512	1.00	2.0000	3.00	4.00	5.00
SOP	500.0	3.37400	0.991004	1.00	2.5000	3.50	4.00	5.00
LOR	500.0	3.48400	0.925450	1.00	3.0000	3.50	4.00	5.00
CGPA	500.0	8.57644	0.604813	6.80	8.1275	8.56	9.04	9.92
Research	500.0	0.56000	0.496884	0.00	0.0000	1.00	1.00	1.00
Chance of Admit	500.0	0.72174	0.141140	0.34	0.6300	0.72	0.82	0.97

Splitting the Independent and dependent variables of the dataframe.

In [80]:
```
data.columns
```

Out[80]:
```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [82]:
```
data.head()
```

Out[82]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

In [53]:
```
x = data.drop('Chance of Admit ', axis=1)
x.head()
```

Out[53]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0

In [86]:
```
y = data['Chance of Admit ']
y.head()
```

Out[86]:
```
0    0.92
1    0.76
2    0.72
3    0.80
4    0.65
Name: Chance of Admit , dtype: float64
```

In [87]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_
```

In [88]:
```
x_train.head()
```

Out[88]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
471	311	103	3	2.0	4.0	8.09	0
26	322	109	5	4.5	3.5	8.80	0
7	308	101	2	3.0	4.0	7.90	0
453	319	103	3	2.5	4.0	8.76	1
108	331	116	5	5.0	5.0	9.38	1

In [89]:
```
y_train.head()
```

Out[89]:
```
471    0.64
26     0.76
7      0.68
453    0.73
108    0.93
Name: Chance of Admit , dtype: float64
```

In [117]:
```
x_test.head()
```

Out[117]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
361	334	116	4	4.0	3.5	9.54	1
73	314	108	4	4.5	4.0	9.04	1
374	315	105	2	2.0	2.5	7.65	0
155	312	109	3	3.0	3.0	8.69	0
104	326	112	3	3.5	3.0	9.05	1

In [118]:
```
y_test.head()
```

Out[118]:
```
361    0.93
73     0.84
374    0.39
155    0.77
104    0.74
Name: Chance of Admit , dtype: float64
```

Standardise the data using the standard scaler

In [90]:
```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
x_train_fit_tf1 = scale.fit_transform(x_train)
x_test_tf1 = scale.transform(x_test)
```

In [91]:
```
x_train_fit_tf1
```

Out[91]:
```
array([[-5.25174731e-01, -7.06985745e-01, -8.28848674e-02, ...,
         5.38819022e-01, -8.35765678e-01, -1.14470294e+00],
       [ 4.77035981e-01,  2.76433873e-01,  1.65251704e+00, ...,
        -1.61323061e-03,  3.63045482e-01, -1.14470294e+00],
       [-7.98504925e-01, -1.03479228e+00, -9.50585823e-01, ...,
         5.38819022e-01, -1.15657430e+00, -1.14470294e+00],
       ...,
       [-1.34516531e+00, -1.36259882e+00, -1.81828678e+00, ...,
        -1.62290999e+00, -2.25407747e+00, -1.14470294e+00],
       [-7.07394861e-01, -3.79179206e-01, -9.50585823e-01, ...,
         5.38819022e-01, -1.52803691e+00, -1.14470294e+00],
       [-2.51844537e-01, -2.15275936e-01, -9.50585823e-01, ...,
        -1.61323061e-03, -5.65611050e-01, -1.14470294e+00]])
```

In [92]:
```
x_test_tf1
```

Out[92]:
```
array([[ 1.57035676e+00,  1.42375676e+00,  7.84816088e-01, ...,
        -1.61323061e-03,  1.61251063e+00,  8.73589088e-01],
       [-2.51844537e-01,  1.12530603e-01,  7.84816088e-01, ...,
         5.38819022e-01,  7.68277423e-01,  8.73589088e-01],
       [-1.60734472e-01, -3.79179206e-01, -9.50585823e-01, ...,
        -1.08247774e+00, -1.57869090e+00, -1.14470294e+00],
       ...,
       [-6.96244077e-02, -7.06985745e-01, -9.50585823e-01, ...,
         1.07925128e+00,  2.61737496e-01, -1.14470294e+00],
       [-6.16284796e-01, -1.36259882e+00, -9.50585823e-01, ...,
        -1.62290999e+00, -2.16965415e+00, -1.14470294e+00],
       [ 2.14856571e-02, -2.15275936e-01, -1.81828678e+00, ...,
        -1.61323061e-03, -1.57869090e+00,  8.73589088e-01]])
```

Train the Model with "Support Vector Regression (SVR)" without Hyper-parameter Tuning

In [97]:
```
svr_model = SVR()
svr_model.fit(x_train_fit_tf1, y_train)
```

Out[97]:
```
SVR()
```

In [98]:
```
y_predict = svr_model.predict(x_test_tf1)
y_predict
```

Out[98]:
```
array([0.86758693, 0.78401064, 0.58098697, 0.70575523, 0.79341922,
       0.84684761, 0.50224061, 0.68176766, 0.7890468 , 0.75740703,
       0.65423979, 0.70427289, 0.65291603, 0.87242604, 0.81192102,
       0.46349708, 0.78825361, 0.59120035, 0.49104457, 0.60340786,
       0.65592674, 0.61635369, 0.6936474 , 0.70124134, 0.72962268,
       0.58562002, 0.87484478, 0.83448662, 0.62548557, 0.72594558,
       0.55523603, 0.71151451, 0.57128054, 0.84876902, 0.62839779,
       0.68891659, 0.50715952, 0.88026673, 0.63306979, 0.68165443,
       0.86381144, 0.56125806, 0.63224004, 0.8569825 , 0.84865294,
       0.55755756, 0.89763628, 0.80955106, 0.73115395, 0.87475322,
       0.83025309, 0.52673652, 0.7129972 , 0.5005013 , 0.88352502,
       0.59844805, 0.85569749, 0.70625272, 0.68184095, 0.4847354 ,
       0.63242747, 0.65850305, 0.57102219, 0.6063961 , 0.45817298,
       0.57186897, 0.86583523, 0.85248887, 0.6354494 , 0.66025349,
       0.61181011, 0.72180601, 0.66233978, 0.56776826, 0.5036567 ,
       0.6899727 , 0.8018803 , 0.86644423, 0.48041644, 0.68114013,
       0.64005184, 0.81452509, 0.64495506, 0.81963739, 0.66339816,
       0.62735888, 0.61514186, 0.6949641 , 0.76524607, 0.65459151,
       0.6948774 , 0.8837818 , 0.86372558, 0.65419964, 0.71353782,
       0.46748871, 0.64975199, 0.67837751, 0.65860157, 0.65038784,
       0.74357748, 0.72858227, 0.65395555, 0.65037173, 0.64374594,
       0.54928438, 0.70417457, 0.78454216, 0.59351846, 0.67208265,
       0.55747195, 0.88258153, 0.82390498, 0.87773364, 0.46744847,
       0.78866063, 0.66337648, 0.86053502, 0.59945371, 0.64912757,
       0.70977167, 0.88032606, 0.69654415, 0.64258023, 0.68723917,
       0.6932172 , 0.59926966, 0.86103228, 0.84216051, 0.48503892,
       0.63455957, 0.66966344, 0.81011649, 0.45284159, 0.74729865,
       0.44980171, 0.80859588, 0.8492926 , 0.65347571, 0.68649545,
       0.66108889, 0.63007374, 0.80847405, 0.49090399, 0.89586325,
       0.65365669, 0.88504634, 0.72806371, 0.63203847, 0.70941942,
       0.83292274, 0.64291636, 0.80870476, 0.68992076, 0.6126307 ,
       0.58962794, 0.75228119, 0.76425607, 0.50989297, 0.56351971,
       0.65411852, 0.75287767, 0.67923785, 0.45954006, 0.59832393])
```

In [99]:
```
svr_model.score(x_train_fit_tf1, y_train)
```

Out[99]:
```
0.7757519357301843
```

In [104]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, r2_score
```

r2 score is used here due to the target is in continuos.

In [108]:
```
a_score = r2_score(y_test, y_predict)
a_score
```

Out[108]:
```
0.7601877598637248
```

The Accuracy without Hyper Parameter Tuning is 76 %

In [110]:
```
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVR(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(x_train_fit_tf1, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.486 total time=
0.0s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.488 total time=
0.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.462 total time=
0.0s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.422 total time=
0.0s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.562 total time=
0.0s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.651 total time=
0.0s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.735 total time=
0.0s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.679 total time=
0.0s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.685 total time=
0.0s
```

In [111]:
```
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
SVR(C=1000, gamma=0.0001)
```

In [113]:
```
grid_predict = grid.predict(x_test_tf1)
grid_predict
```

Out[113]:
```
array([0.89084316, 0.77020419, 0.56031344, 0.69786896, 0.80240814,
       0.83231769, 0.47294014, 0.62828712, 0.80696245, 0.7817148 ,
       0.70447524, 0.71478402, 0.61857975, 0.89304663, 0.80886473,
       0.49137979, 0.79264935, 0.57737141, 0.52267001, 0.54580513,
       0.65225706, 0.51771679, 0.69704342, 0.77169561, 0.76507901,
       0.58454293, 0.91745165, 0.81686705, 0.61789398, 0.71717821,
       0.53563798, 0.72034298, 0.53960682, 0.83646701, 0.63113431,
       0.71805788, 0.54604628, 0.92199689, 0.62526755, 0.70397418,
       0.93111204, 0.55364477, 0.65183573, 0.83103647, 0.89768849,
       0.56555204, 0.93109682, 0.8081616 , 0.75649911, 0.89076788,
       0.85059507, 0.54852083, 0.66883774, 0.5085655 , 0.91511985,
       0.56900956, 0.92835111, 0.72914565, 0.64993998, 0.47888246,
       0.6069931 , 0.66523689, 0.58790053, 0.5549128 , 0.42170428,
       0.58521517, 0.84169483, 0.85855745, 0.6366739 , 0.68677191,
       0.60066069, 0.7667449 , 0.67134282, 0.55352271, 0.54648966,
       0.64671887, 0.81863966, 0.84100095, 0.52305934, 0.58546939,
       0.74851021, 0.81155967, 0.5938901 , 0.81625352, 0.70449749,
       0.65163307, 0.59050691, 0.71061155, 0.76235822, 0.65182514,
       0.71875244, 0.88516652, 0.87144527, 0.63065794, 0.77442957,
       0.41811861, 0.66850269, 0.77654886, 0.73758289, 0.63159806,
       0.77720832, 0.72037705, 0.62599573, 0.66258773, 0.63733017,
       0.51825034, 0.71954691, 0.78096002, 0.62604962, 0.65195428,
       0.62504848, 0.91738249, 0.81716895, 0.93196616, 0.44240478,
       0.8053495 , 0.65980295, 0.84743415, 0.46960033, 0.68196357,
       0.6940359 , 0.86726699, 0.70309195, 0.72126982, 0.67195742,
       0.69250536, 0.54637887, 0.95727813, 0.83374574, 0.49479757,
       0.69087128, 0.71875975, 0.80805115, 0.49941829, 0.81793111,
       0.4894271 , 0.79385   , 0.92934939, 0.64217857, 0.65605998,
       0.78680743, 0.61549683, 0.7900748 , 0.50117029, 0.91777798,
       0.64758397, 0.9300943 , 0.76055809, 0.61949108, 0.69706383,
       0.84433901, 0.64054984, 0.79471592, 0.70137317, 0.5748844 ,
       0.59800229, 0.73566877, 0.77854278, 0.4834825 , 0.58658212,
       0.64109156, 0.74889631, 0.70649057, 0.47076258, 0.61059363])
```

In [114]:
```
a_score_g = r2_score(y_test, grid_predict)
a_score_g
```

Out[114]:
```
0.8028692319045374
```

After tuning the Hyper Parameter using GridSearch_CV

The Accuracy has been Improve from 76 % to 80 %.

In [ ]:
```

```