# Regression Models Report

## Improving machine learning skills by developing regression models

Student:

Anurag Banger

Student Id:

20642433

# Introduction

Regression models are developed to analysis the data. We train our model so that it gives better output for our test data. In this report we will be developing 3 regression models: linear regression model, support vector regression model and k-nearest neighbour regression model. There are two types of machine learning algorithms: Regression which predicts continuous value outputs and Classification which predicts discrete outputs.

```
#Import the necessary libraries
from pandas.plotting import scatter_matrix
from pandas import read_csv
import pandas as pd
import numpy as np
from matplotlib import pyplot
import seaborn as seabornInstance

#Reading the data
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataset = read_csv(filename, delim_whitespace=True, names=names)
```

By “print(dataset.shape)” we can get the shape of the dataset as (506, 14).

The info of the dataset can be checked by “dataset,info()”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

Below is the data description of the “housing.csv” data that we are using for this report.

| | CRIM | ZN | INDUS | CHAS | NOX | RM |
|---|---|---|---|---|---|---|
| count | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 |
| mean | 3.613524 | 11.363636 | 11.136779 | 0.069170 | 0.554695 | 6.284634 |
| std | 8.601545 | 23.322453 | 6.860353 | 0.253994 | 0.115878 | 0.702617 |
| min | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 |
| 25% | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 |
| 50% | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 |
| 75% | 3.677082 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 |
| max | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 |

| | AGE | DIS | RAD | TAX | PTRATIO | B |
|---|---|---|---|---|---|---|
| count | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 |
| mean | 68.574901 | 3.795043 | 9.549407 | 408.237154 | 18.455534 | 356.674032 |
| std | 28.148861 | 2.105710 | 8.707259 | 168.537116 | 2.164946 | 91.294864 |
| min | 2.900000 | 1.129600 | 1.000000 | 187.000000 | 12.600000 | 0.320000 |
| 25% | 45.025000 | 2.100175 | 4.000000 | 279.000000 | 17.400000 | 375.377500 |
| 50% | 77.500000 | 3.207450 | 5.000000 | 330.000000 | 19.050000 | 391.440000 |
| 75% | 94.075000 | 5.188425 | 24.000000 | 666.000000 | 20.200000 | 396.225000 |
| max | 100.000000 | 12.126500 | 24.000000 | 711.000000 | 22.000000 | 396.900000 |

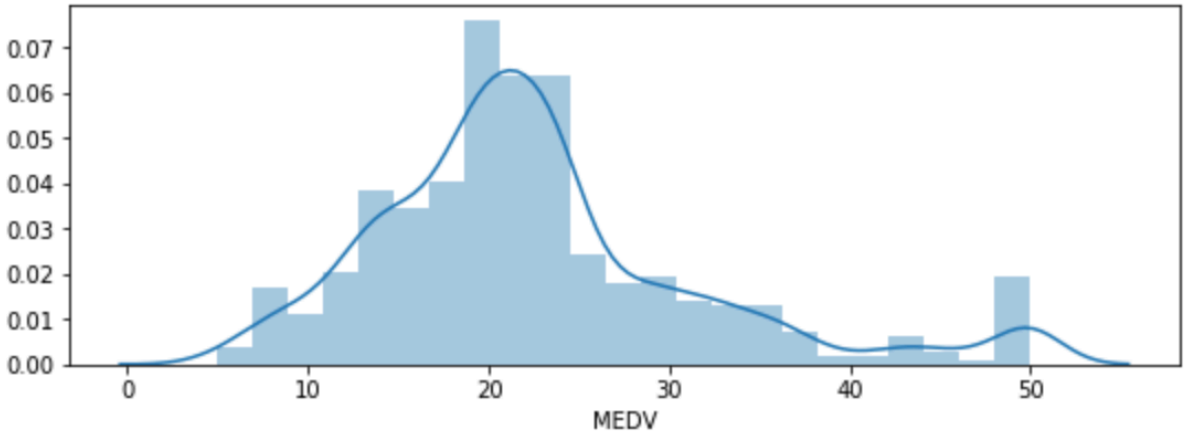
| | LSTAT | MEDV |
|---|---|---|
| count | 506.000000 | 506.000000 |
| mean | 12.653063 | 22.532806 |
| std | 7.141062 | 9.197104 |
| min | 1.730000 | 5.000000 |
| 25% | 6.950000 | 17.025000 |
| 50% | 11.360000 | 21.200000 |
| 75% | 16.955000 | 25.000000 |
| max | 37.970000 | 50.000000 |

Correlation data for the dataset is

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE |
|---|---|---|---|---|---|---|---|
| CRIM | 1.000000 | -0.200469 | 0.406583 | -0.055892 | 0.420972 | -0.219247 | 0.352734 |
| ZN | -0.200469 | 1.000000 | -0.533828 | -0.042697 | -0.516604 | 0.311991 | -0.569537 |
| INDUS | 0.406583 | -0.533828 | 1.000000 | 0.062938 | 0.763651 | -0.391676 | 0.644779 |
| CHAS | -0.055892 | -0.042697 | 0.062938 | 1.000000 | 0.091203 | 0.091251 | 0.086518 |
| NOX | 0.420972 | -0.516604 | 0.763651 | 0.091203 | 1.000000 | -0.302188 | 0.731470 |
| RM | -0.219247 | 0.311991 | -0.391676 | 0.091251 | -0.302188 | 1.000000 | -0.240265 |
| AGE | 0.352734 | -0.569537 | 0.644779 | 0.086518 | 0.731470 | -0.240265 | 1.000000 |
| DIS | -0.379670 | 0.664408 | -0.708027 | -0.099176 | -0.769230 | 0.205246 | -0.747881 |
| RAD | 0.625505 | -0.311948 | 0.595129 | -0.007368 | 0.611441 | -0.209847 | 0.456022 |
| TAX | 0.582764 | -0.314563 | 0.720760 | -0.035587 | 0.668023 | -0.292048 | 0.506456 |
| PTRATIO | 0.289946 | -0.391679 | 0.383248 | -0.121515 | 0.188933 | -0.355501 | 0.261515 |
| B | -0.385064 | 0.175520 | -0.356977 | 0.048788 | -0.380051 | 0.128069 | -0.273534 |
| LSTAT | 0.455621 | -0.412995 | 0.603800 | -0.053929 | 0.590879 | -0.613808 | 0.602339 |
| MEDV | -0.388305 | 0.360445 | -0.483725 | 0.175260 | -0.427321 | 0.695360 | -0.376955 |

| | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|
| CRIM | -0.379670 | 0.625505 | 0.582764 | 0.289946 | -0.385064 | 0.455621 | -0.388305 |
| ZN | 0.664408 | -0.311948 | -0.314563 | -0.391679 | 0.175520 | -0.412995 | 0.360445 |
| INDUS | -0.708027 | 0.595129 | 0.720760 | 0.383248 | -0.356977 | 0.603800 | -0.483725 |
| CHAS | -0.099176 | -0.007368 | -0.035587 | -0.121515 | 0.048788 | -0.053929 | 0.175260 |
| NOX | -0.769230 | 0.611441 | 0.668023 | 0.188933 | -0.380051 | 0.590879 | -0.427321 |
| RM | 0.205246 | -0.209847 | -0.292048 | -0.355501 | 0.128069 | -0.613808 | 0.695360 |
| AGE | -0.747881 | 0.456022 | 0.506456 | 0.261515 | -0.273534 | 0.602339 | -0.376955 |
| DIS | 1.000000 | -0.494588 | -0.534432 | -0.232471 | 0.291512 | -0.496996 | 0.249929 |
| RAD | -0.494588 | 1.000000 | 0.910228 | 0.464741 | -0.444413 | 0.488676 | -0.381626 |
| TAX | -0.534432 | 0.910228 | 1.000000 | 0.460853 | -0.441808 | 0.543993 | -0.468536 |
| PTRATIO | -0.232471 | 0.464741 | 0.460853 | 1.000000 | -0.177383 | 0.374044 | -0.507787 |
| B | 0.291512 | -0.444413 | -0.441808 | -0.177383 | 1.000000 | -0.366087 | 0.333461 |
| LSTAT | -0.496996 | 0.488676 | 0.543993 | 0.374044 | -0.366087 | 1.000000 | -0.737663 |
| MEDV | 0.249929 | -0.381626 | -0.468536 | -0.507787 | 0.333461 | -0.737663 | 1.000000 |

```
pyplot.figure(figsize=(9,3))
pyplot.tight_layout()
seabornInstance.distplot(dataset['MEDV'])
```




Our regression model should predict output similar to the above graph.

```
#Train Test Split
from sklearn.model_selection import train_test_split
array = dataset.values
X = array[:,0:13]
Y = array[:,13]
validation_size = 0.20
seed = 85
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=validation_size, random_state=seed)
```

Using the above code, we are splitting the dataset into train and test at the random state of 85. In the "*train_test_split*" function initially I used the random state as 50, I was getting a high error. So, I tried increased the random state to 70, the error reduced and the r2_score increased. However, when I increase it to 90 the error increase and the r2_scrore decreased. Finally, I Settled for 85 as the best random state for all the below regression models.

The final step is to evaluate the performance of the algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

1. **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - y_j|$$

2. **Mean Squared Error** (MSE) is the mean of the squared errors and is calculated as:

$$MSE = \frac{1}{N}\sum_{i}^{n}(Y_i - y_i)^2$$

3. **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

# Linear Regression Model

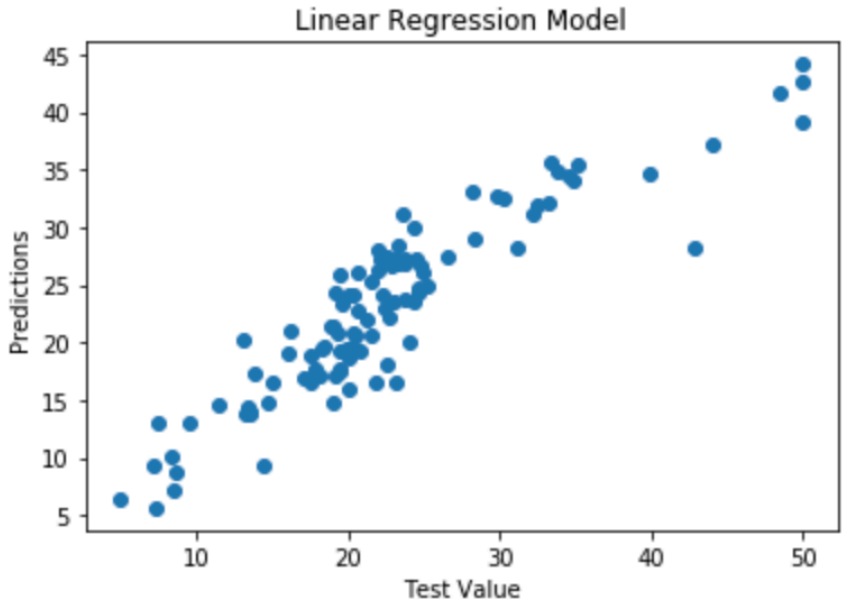
Linear regression performs the tasks to predict a dependent variable (Y) using the given variable (X). We will be using multiple linear regression for this problem. For multivariable linear regression, the linear the regression model has to find the most optimal coefficients for all the attributes. From the table 1, we can that for a unit increase in "*nitric oxides concentration (NOX)",* there is a decrease of 20.38 units in the median value and a unit decrease in "*average number of rooms per dwelling (RM)*" there is an increase of 3.14 units in the median value. We can check the difference between the actual and predicted value from the table 2 and the graph of actual and predicted.

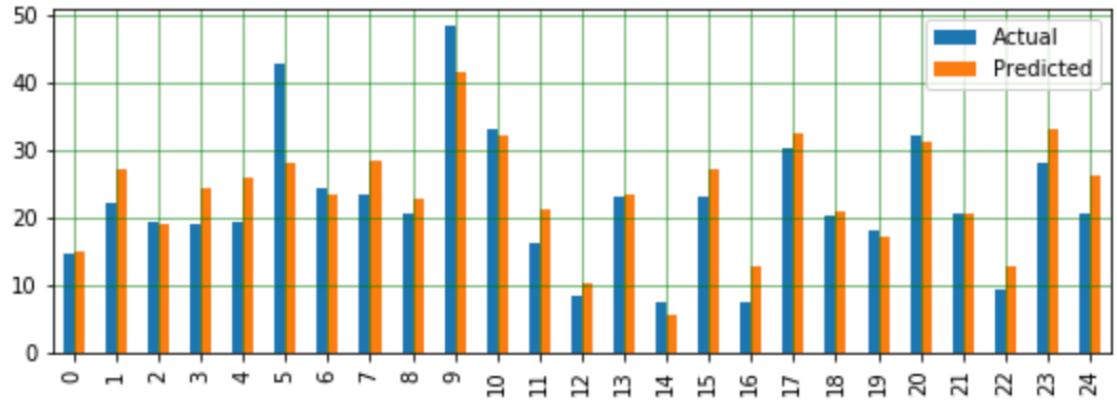
Table 1

| | Coefficient |
|---|---|
| CRIM | -0.110900 |
| ZN | 0.048214 |
| INDUS | 0.002390 |
| CHAS | 2.669476 |
| NOX | -20.384569 |
| RM | 3.146873 |
| AGE | 0.012821 |
| DIS | -1.594610 |
| RAD | 0.319752 |
| TAX | -0.012424 |
| PTRATIO | -0.996783 |
| B | 0.008374 |
| LSTAT | -0.571749 |

Table 2

| | Actual | Predicted |
|---|---|---|
| 0 | 14.8 | 14.868280 |
| 1 | 22.1 | 27.281833 |
| 2 | 19.5 | 19.213757 |
| 3 | 19.2 | 24.342587 |
| 4 | 19.4 | 25.964260 |
| 5 | 42.8 | 28.179739 |
| 6 | 24.4 | 23.538190 |
| 7 | 23.3 | 28.464132 |
| 8 | 20.6 | 22.702733 |
| 9 | 48.5 | 41.669042 |
| 10 | 33.2 | 32.183417 |
| 11 | 16.2 | 21.107817 |
| 12 | 8.3 | 10.196391 |
| 13 | 23.0 | 23.472528 |
| 14 | 7.4 | 5.663803 |
| 15 | 23.2 | 27.166357 |
| 16 | 7.5 | 12.942476 |
| 17 | 30.3 | 32.473298 |
| 18 | 20.4 | 20.817842 |
| 19 | 18.1 | 17.062482 |
| 20 | 32.2 | 31.093483 |
| 21 | 20.5 | 20.646620 |
| 22 | 9.5 | 12.973219 |
| 23 | 28.2 | 33.143395 |
| 24 | 20.7 | 26.111568 |




```
regressor = LinearRegression(normalize=True)
```

**Output for linear regression model:**

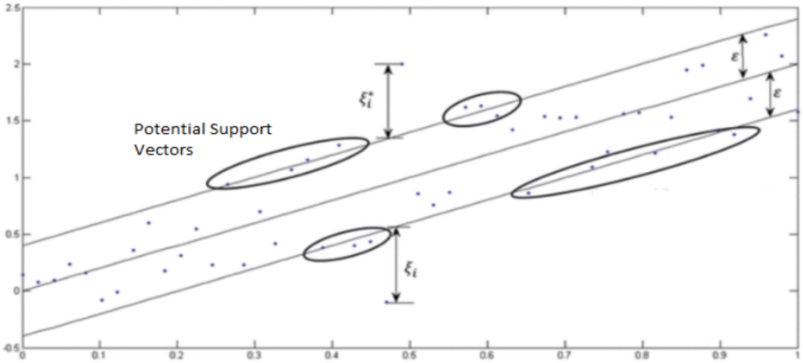
```
Mean Absolute Error: 2.7770165268624316
Mean Squared Error: 14.053146682822419
Root Mean Squared Error: 3.74875268360323
r2_score: 0.8300388580559296
explained variance Score: 0.8328922853022158
```

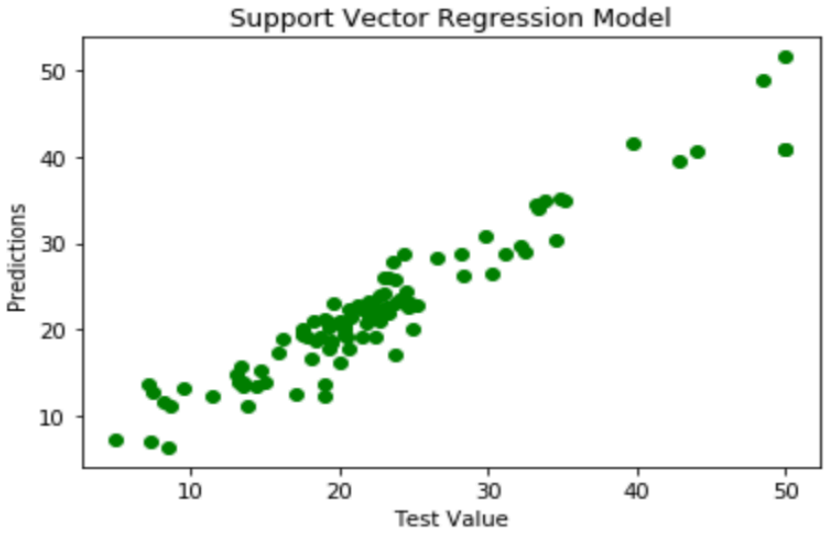
For this model I tuned the '*random_state'* to 85 to get such low error.

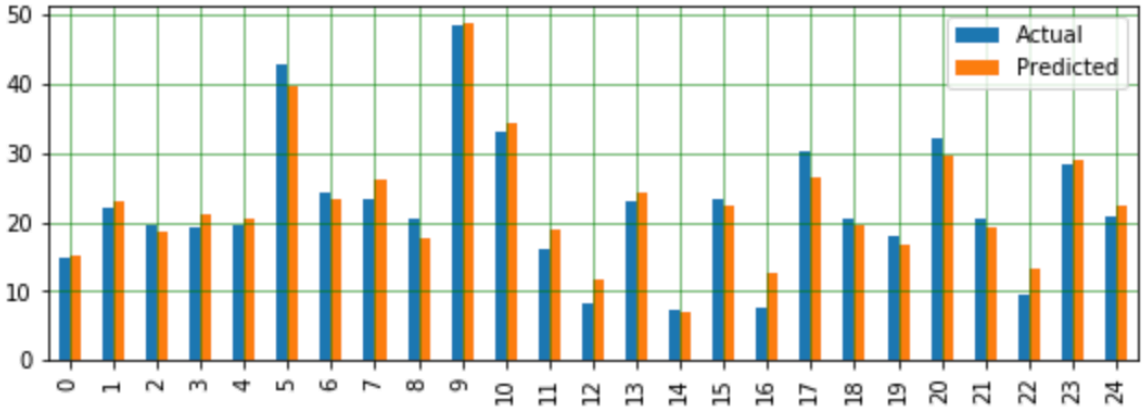
# SVM Model

Support Vector regression is a type of Support vector machine that supports linear and non-linear regression. As it seems in the below graph, the mission is to fit as many instances as possible between the lines while limiting the margin violations. The violation concept in this example represents as ε (epsilon). I have used the "*StandardScaler*" function to normalize the data so that it can be easier for the model to read the data and make better predictions.

To build a SVR model we have to first choose a kernel and parameters. I trained my model using ‘rbf’, ‘linear’ and ‘poly’ kernel. I found out the kernel = ‘rbf’ gives the best output. Changing the epsilon and C value will further give a more optimised output.



| | Actual | Predicted |
|---|---|---|
| 0 | 14.8 | 15.282607 |
| 1 | 22.1 | 23.068043 |
| 2 | 19.5 | 18.727181 |
| 3 | 19.2 | 21.064892 |
| 4 | 19.4 | 20.386563 |
| 5 | 42.8 | 39.541974 |
| 6 | 24.4 | 23.299517 |
| 7 | 23.3 | 26.069507 |
| 8 | 20.6 | 17.798168 |
| 9 | 48.5 | 48.899721 |
| 10 | 33.2 | 34.391573 |
| 11 | 16.2 | 18.880577 |
| 12 | 8.3 | 11.528066 |
| 13 | 23.0 | 24.158698 |
| 14 | 7.4 | 6.932648 |
| 15 | 23.2 | 22.471696 |
| 16 | 7.5 | 12.670828 |
| 17 | 30.3 | 26.386681 |
| 18 | 20.4 | 19.485129 |
| 19 | 18.1 | 16.649052 |
| 20 | 32.2 | 29.586668 |
| 21 | 20.5 | 19.164047 |
| 22 | 9.5 | 13.286589 |
| 23 | 28.2 | 28.842379 |
| 24 | 20.7 | 22.460207 |

```
model = SVR(kernel='rbf', C=100)
```

**Output for SVR model:**

```
Mean Absolute Error: 2.045683998294405
Mean Squared Error: 7.432523737676976
Root Mean Squared Error: 2.7262655295618172
r2_score: 0.910109796012726
explained variance Score: 0.9107069269827175
```

# K–Nearest Neighbour Model

"*KNeighborsRegressor*" implements learning based on the k nearest neighbours of each query point, where k is an integer value specified by the user. KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition. Normalization of the data in this model has been done using "*StandardScaler*". Cross-validation is when the dataset is randomly split up into 'k' groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group has been used as the test set. Below are the cross validation scores for the KNN model. In our model we are using a 6 fold cross validation to get the best output. The train-test-split method we used in earlier is called 'holdout'. Cross-validation is better than using the holdout method because the holdout method score is dependent on how the data is split into train and test sets. Cross-validation gives the model an opportunity to test on multiple splits so we can get a better idea on how the model will perform on unseen data.

```
[0.82105336 0.6195712  0.74470466 0.78805673 0.78792551 0.80417367]
cv_scores mean:{} 0.7609141886383952
```

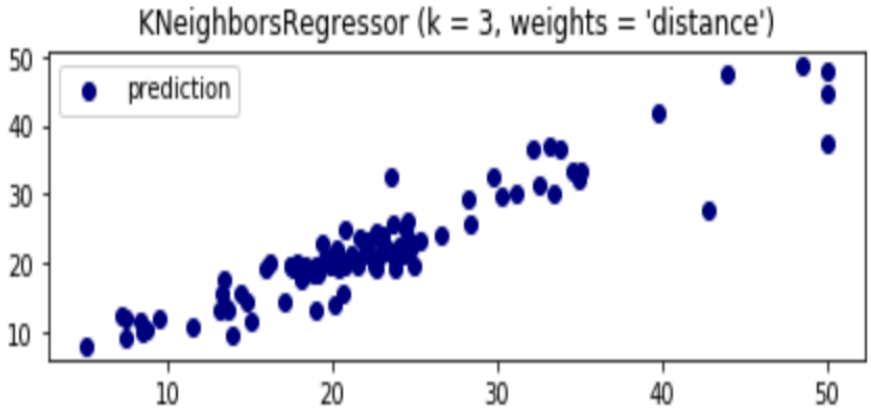
Hypertuning parameters is when you go through a process to find the optimal parameters for your model to improve accuracy. In our case, we will use GridSearchCV to find the optimal value for 'n_neighbors'. GridSearchCV works by training our model multiple times on a range of parameters that we specify. That way, we can test our model with each parameter and figure out the optimal values to get the best accuracy results. The best params we get it for the n_neighbour of 3. KNeighborsRegressor has many parameters out of which I tuned weights and algorithm. I encountered 2 types of weights: uniform and distance, distance weight had lesser error and higher r2_score compared to uniform weight for nearest neighbor = 3. The algorithm *'brute'* is very efficient for a small data sample. However, as the number of samples increases the efficiency decreases. Algorithm '*kd_tree*' is very fast because partitioning is performed only along the data axes. The algorithm '*ball_tree*' is a single distance calculation between a test point and the centroid is sufficient to determine a lower and upper bound on the distance to all points within the node. Because of the spherical geometry of the ball tree nodes, it can out-

perform a *KD-tree* in high dimensions, though the actual performance is highly dependent on the structure of the training data.

Grid search using cross validation:

```
from sklearn.model_selection import GridSearchCV
knn2 = KNeighborsRegressor()
param_grid = {'n_neighbors': np.arange(1, 25)}
#use gridsearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn2, param_grid, cv=6)
#fit model to data
knn_gscv.fit(X_train_std, Y_train)
```

| | Actual | Predicted |
|---|---|---|
| 0 | 14.8 | 14.429577 |
| 1 | 22.1 | 23.296695 |
| 2 | 19.5 | 19.948124 |
| 3 | 19.2 | 19.655173 |
| 4 | 19.4 | 22.990306 |
| 5 | 42.8 | 27.649959 |
| 6 | 24.4 | 24.879748 |
| 7 | 23.3 | 22.058364 |
| 8 | 20.6 | 15.507271 |
| 9 | 48.5 | 48.910176 |
| 10 | 33.2 | 37.118727 |
| 11 | 16.2 | 20.022242 |
| 12 | 8.3 | 11.638765 |
| 13 | 23.0 | 23.889208 |
| 14 | 7.4 | 9.239357 |
| 15 | 23.2 | 21.569757 |
| 16 | 7.5 | 12.004898 |
| 17 | 30.3 | 29.872230 |
| 18 | 20.4 | 19.371515 |
| 19 | 18.1 | 17.668061 |
| 20 | 32.2 | 36.660045 |
| 21 | 20.5 | 19.620360 |
| 22 | 9.5 | 12.036435 |
| 23 | 28.2 | 29.301961 |
| 24 | 20.7 | 25.033085 |




```
knn = KNeighborsRegressor(3, weights='distance')
```

**Output for KNN:**

```
Mean Absolute Error: 2.277490832961668
Mean Squared Error: 10.503144584772654
Root Mean Squared Error: 3.240855532845093
r2_score: 0.8729731861538417
explained variance Score: 0.8736631499908325
```

# Conclusion

In the above 3 models general 6 Steps are being performed: importing libraries, loading the dat asets, dividing the train and test datasets, feature scaling, fitting the support vector regression m odel, predicting the result and visualising the support vector results by comparing it with the tes t results. By observing the above regression models we can conclude that SVR model with kern el as 'rbf' and C as 100 has the best predictions. The output of SVR shows that mean absolute e rror, mean squared error and root mean squared error are lowest and r2_score and explained va riance score are the highest among all the 3 models. We can see that the value of root mean squ ared error is 2.726 which is greater than 10% of the mean value which is 22.533. This means t hat our algorithm is not very accurate but can still make reasonably good predictions.

# Reference


- https://medium.com/pursuitnotes/support-vector-regression-in-6-steps-with-python-c4569acd062d
- https://scikit-learn.org/stable/auto_examples/neighbors/plot_regression.html
- https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f
- https://scikit-learn.org/stable/modules/neighbors.html
- https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/