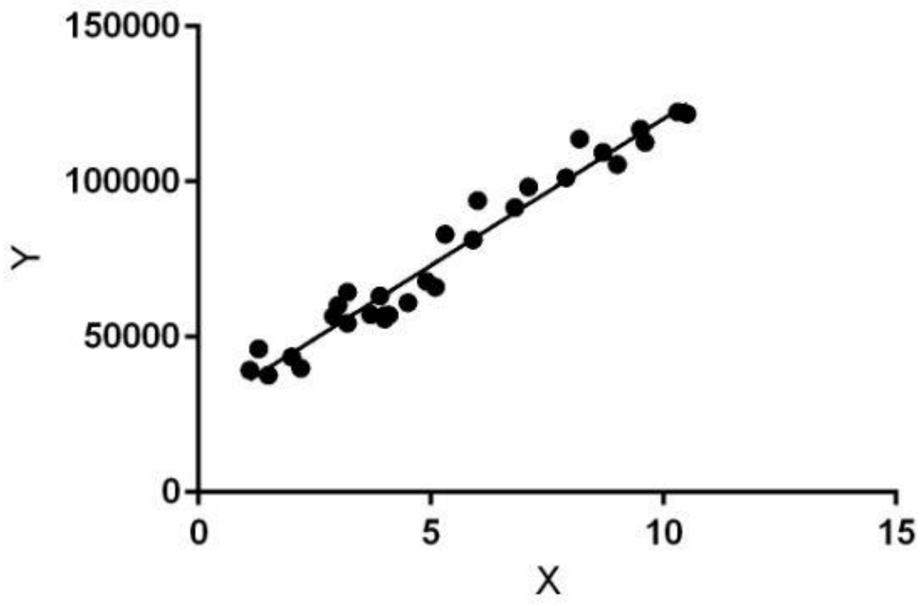
Machin Learning, Regression Models for accurately predicting median value of Boston houses in any given suburb.

By

Darshan H S
Student Id:20669886

LINEAR REGRESSION MODEL

- Linear Regression is a one of the Supervised Machin learning algorithms, it used to predict dependent value y based on the given independent variable x.



In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Reference Image: www.geeksforgeeks.org

Equation Linear Regression model

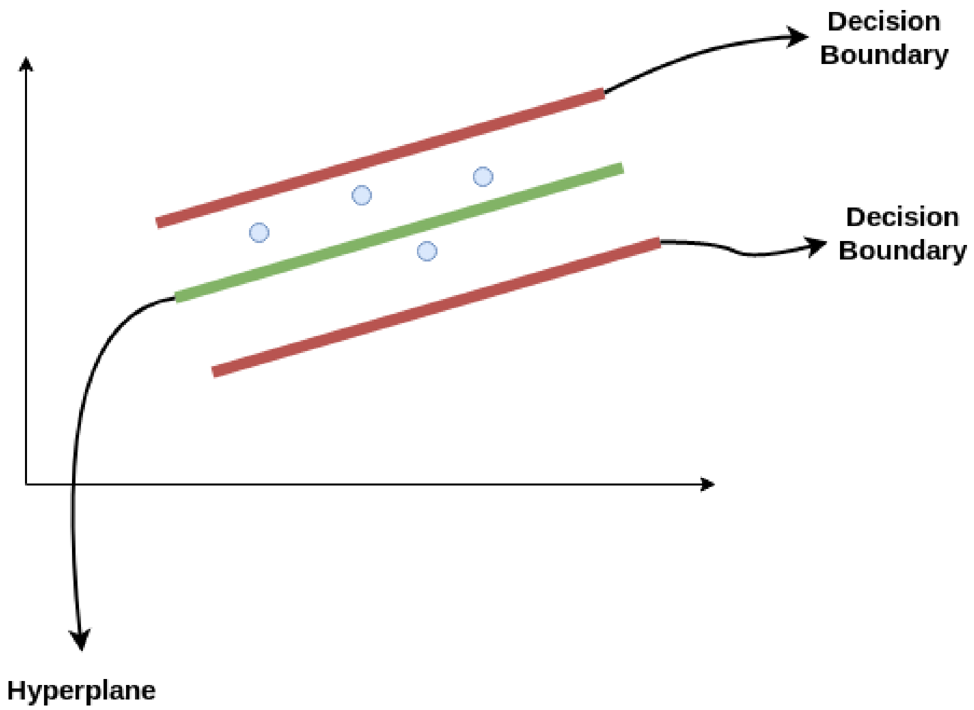
$y = \beta0 + \beta1x + e;$

- y is the output variable
- x is the input variable
- $\beta0$ is the intercept (the value of y when $x=0$)

- β1 is the coefficient for x and the slope of the regression line (“the average increase in Y associated with a one-unit increase in X”)
- e is the error term

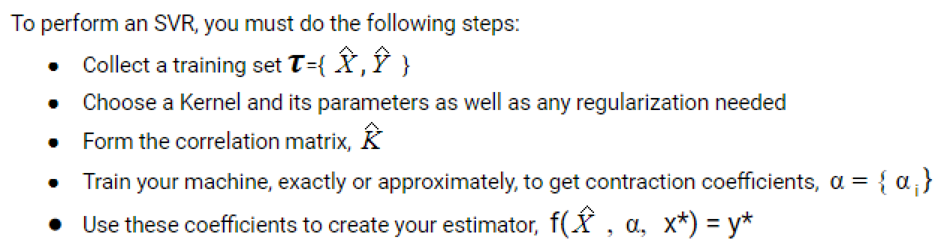
SUPPORT VECTOR MACHINE(SVMR)

Support Vector Regression (SVR) is quite different than other Regression models. It uses the Support Vector Machine (SVM, a classification algorithm) algorithm to predict a continuous variable. While other linear regression models try to minimize the error between the predicted and the actual value, Support Vector Regression tries to fit the best line within a predefined or threshold error value.



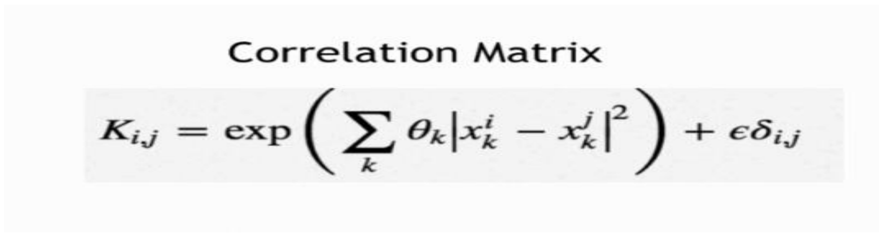
- Consider these two red lines as the decision boundary and the green line as the hyperplane. **Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.

•


To perform an SVR, you must do the following steps:

- Collect a training set $\boldsymbol{\tau}$={ $\hat{X}$, $\hat{Y}$ }
- Choose a Kernel and its parameters as well as any regularization needed
- Form the correlation matrix, $\hat{K}$
- Train your machine, exactly or approximately, to get contraction coefficients, $\alpha = \{ \alpha_i \}$
- Use these coefficients to create your estimator, $f(\hat{X}, \alpha, x^*) = y^*$

•


Correlation Matrix

$$K_{i,j} = \exp\left(\sum_k \theta_k |x_k^i - x_k^j|^2\right) + \epsilon\delta_{i,j}$$

SVM KERNELS

- **Linear Kernel** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.
- **Polynomial Kernel** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.
- **Radial Basis Function Kernel** the Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

K-NEAREST NEIGHBORS

- KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution.
- In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real-world datasets do not follow mathematical theoretical assumptions.
- Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier.

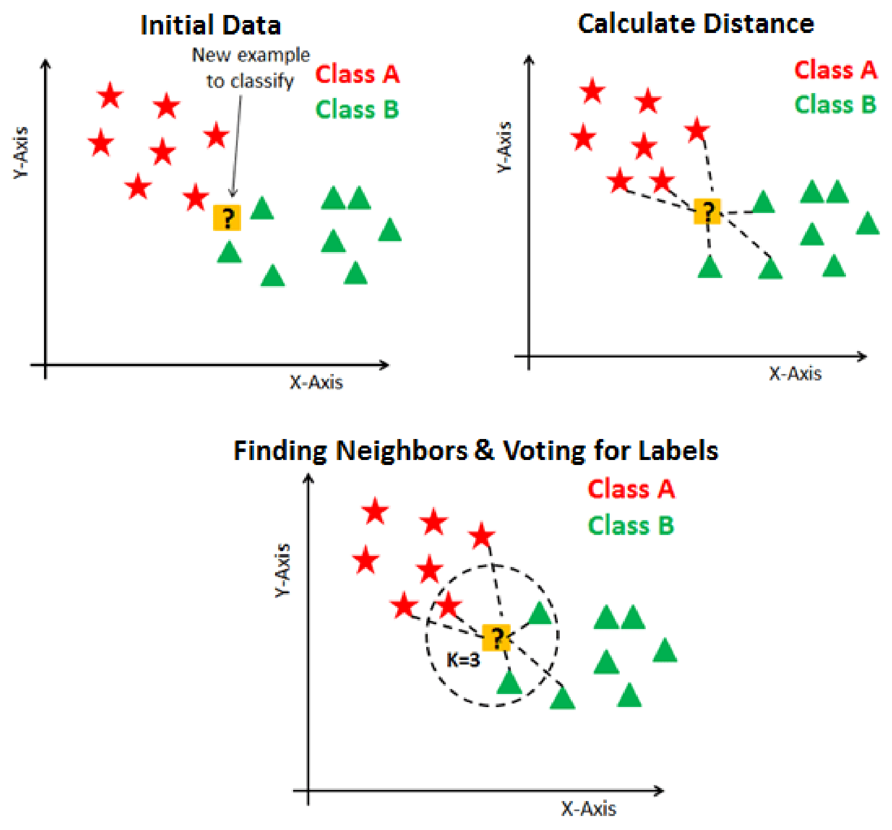
- Costly testing phase means time and memory. In the worst case, KNN needs more time to scan all data points and scanning all data points will require more memory for storing training data.

KNN has the following basic steps:

- Calculate distance

- Find closest neighbours

- Vote for labels



K nearest neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions).

Algorithm

A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance functions as KNN classification.

Distance functions

Euclidean	$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$
Manhattan	$\sum_{i=1}^{k}\lvert x_i - y_i\rvert$
Minkowski	$\left(\sum_{i=1}^{k}(\lvert x_i - y_i\rvert)^q\right)^{1/q}$

The above three distance measures are only valid for continuous variables. In the case of categorical variables, you must use the Hamming distance, which is a measure of the number of instances in which corresponding symbols are different in two strings of equal length.

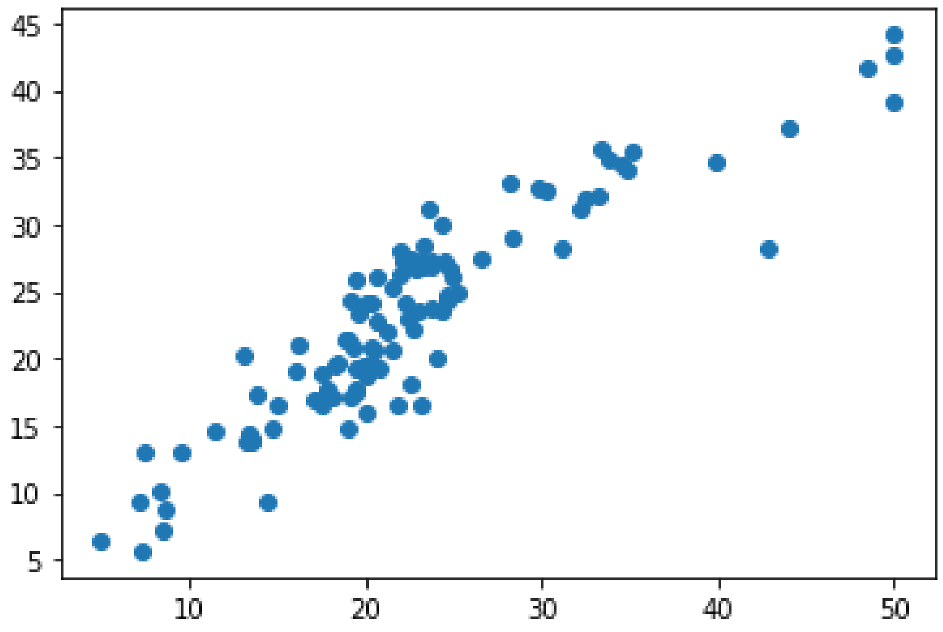
Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise; however, the compromise is that the distinct boundaries within the feature space are blurred. Cross-validation is another way to retrospectively determine a good K value by using an independent data set to validate your K value. The optimal K for most datasets is 10 or more. That produces much better results than 1-NN.

X_train, X_test, y_train, y_test = train_test_split (X, Y, test size=0.20, random state=85)

While training the data I absorbed that changing random state we get better results, specifically for random state = 85 found good results for every model.

Model 1: ***Linear Regression***

Below is the Plots for Linear Regression



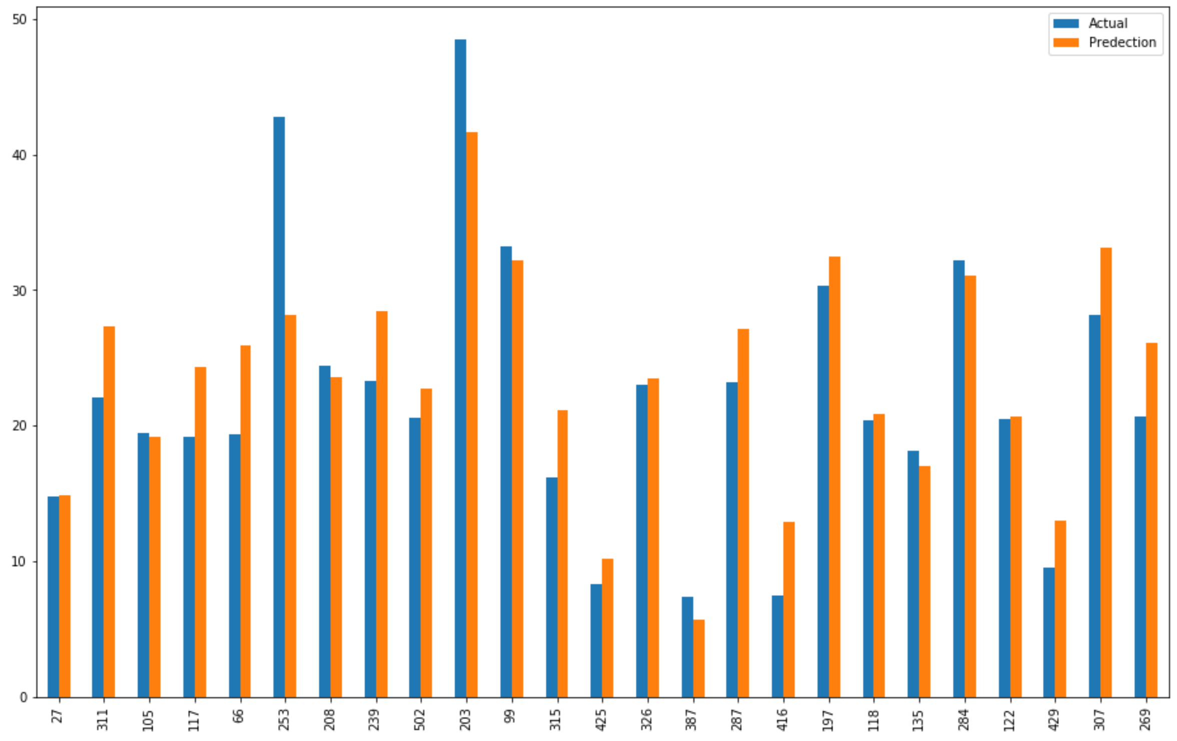
By above graph we can see that its very good model reaching approximately tight straight line with minimum error.

For Liner Regression Model

Tried with all the different Parameters, by changing the below parameters with different values but the results are same.

LinearRegression (fit_intercept=True, normalize=False, copy_X=True)

Below is the Example bar for predict and test data



mean_absolute_error(y_test,y_test_pred)

: 2.77701652686242

mean_squared_error(y_test,y_test_pred)

: 14.053146682822302

r2_score(y_train,y_train_pred)

: 830038858055931

explained_variance_score(y_train,y_train_pred)

:0.8328922853022179

For the Next two models I will be using StandardScaler and Grid search method to Optimize and get accurate prediction.

StandardScaler: The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.In case of multivariate data, this is done feature-wise (in other words independently for each column of the data).
Given the distribution of the data, each value in the dataset will have the mean value subtracted, and then divided by the standard deviation of the whole dataset (or feature in the multivariate case).

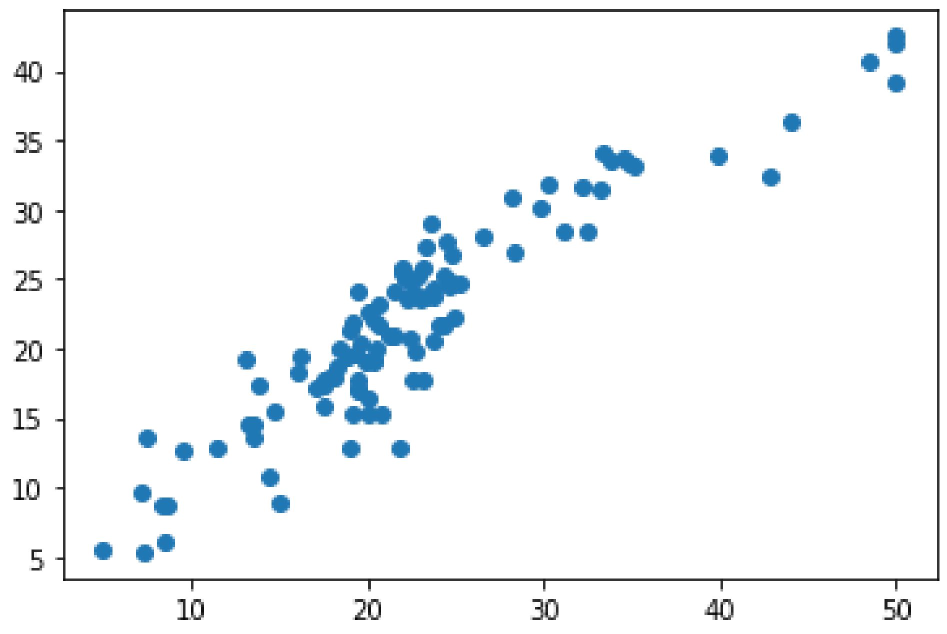
class sklearn.preprocessing.**StandardScaler**(*, copy=True, with_mean=True, with_std=True)

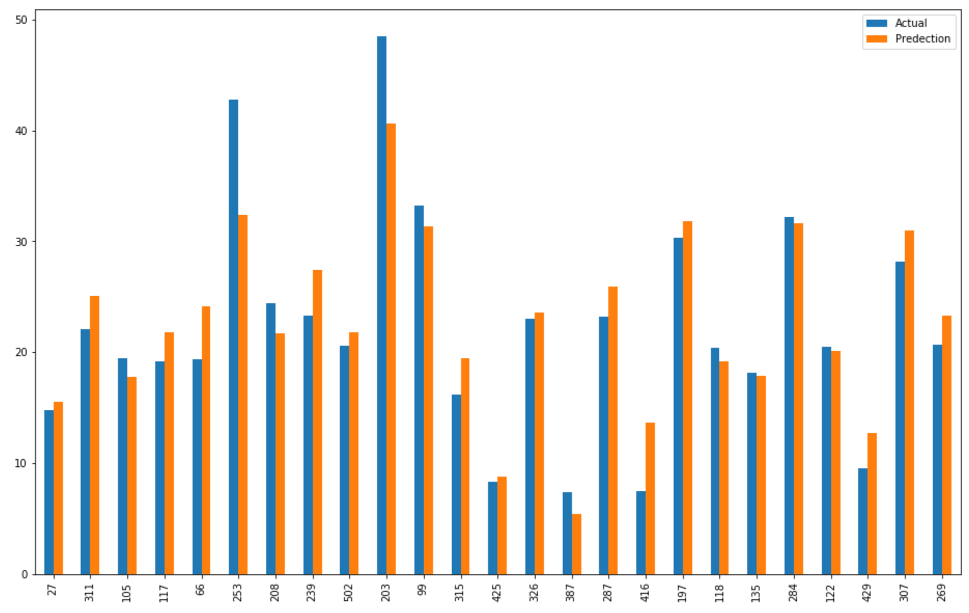
Grid search method: Grid-search is used to find the optimal hyperparameters of a model which results in the most ‘accurate’ predictions.

class sklearn.model_selection.**GridSearchCV**(estimator, param_grid, *, scoring=None, n_jobs=None, iid='deprecated', refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False)

Model 2: ***Support vector Machin for Regression***

svr = SVR(kernel='linear ',C = 1000) : basic model with kernel linear , were C =1000 Regularization parameter below is Linear graph , bar graph and results.





We can see from the above graph the prediction value and test values are not to the closest, and scatter graph is linear with best fit but below results we can that accuracy is very low.

mean_absolute_error(y_test,y_test_pred)

: 2.563169989735859

mean_squared_error(y_test,y_test_pred)

: 12.106495403700405

r2_score(y_train,y_train_pred)

: 0.6763477713690073

explained_variance_score(y_train,y_train_pred)

: 0.6884477799300694

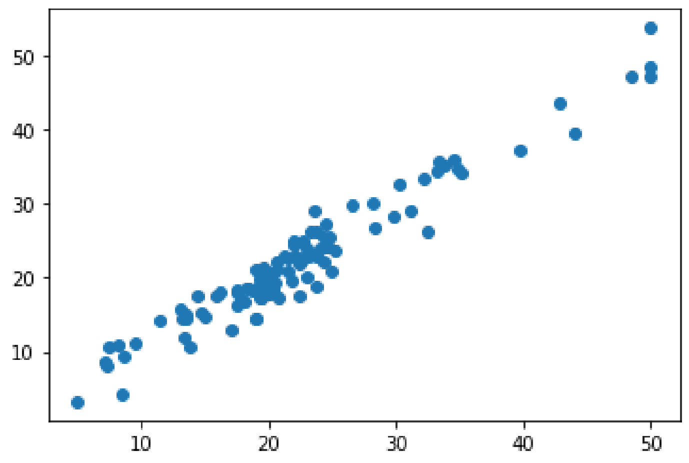
Second model of SVMR is

from sklearn.svm import SVR

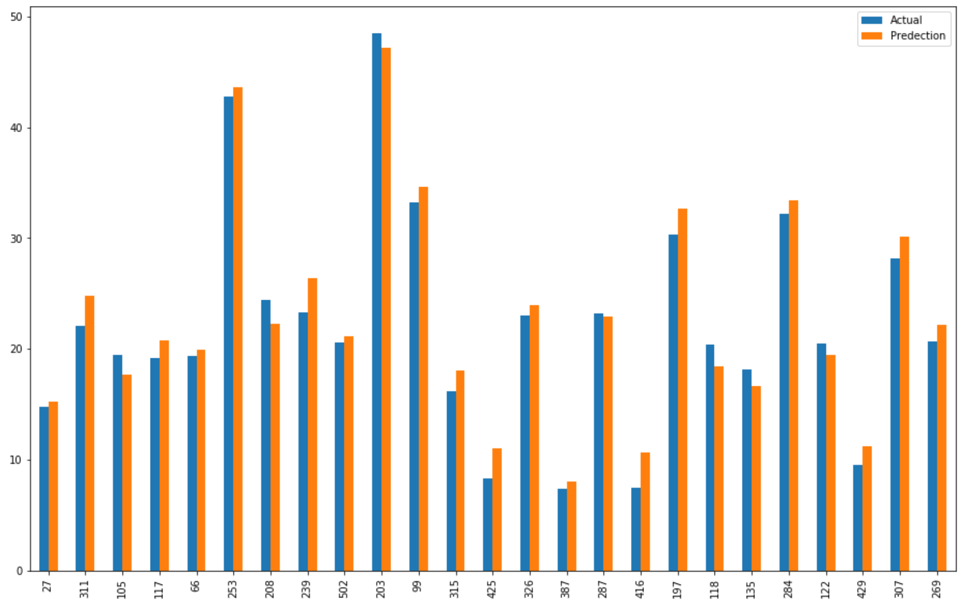
svr = SVR (kernel = 'rbf', C = 100, gamma=0.01)

kernel = rbf, C=100, gamma=0.01 is the best parameters, we good very good accuracy rate minimum mean absolute error.

below is the linear and Bar graphs



We can see the best fit and its almost the straight , with minimum error.



predictive percentage of the model is 93%

mean_absolute_error (y_test, y_test_pred)

: 1.830769318985559

mean_squared_error (y_test, y_test_pred)

: 5.117728962560878

r2_score (y_train, y_train_pred)

: 0.9381053170319292

explained_variance_score (y_train, y_train_pred)

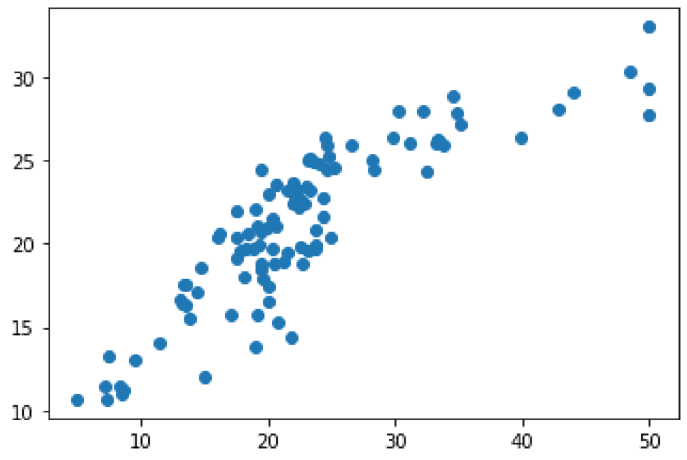
: 0.9382318306072851

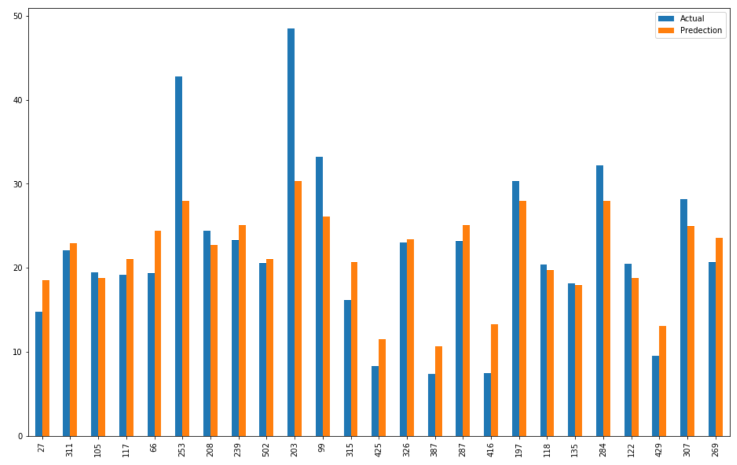
Third Model for SVMR

svr = SVR (kernel='sigmoid' C = 10, gamma=0.001)

This model used sigmoid kernel and C =10 (with grid search)

This model does not suite with this data set. It has lower accuracy and prediction rate.





Above two graphs show that linear is not a best fit and prediction rates are very low

mean_absolute_error (y_test, y_test_pred)

: 3.8100829952728246

mean_squared_error (y_test, y_test_pred)

: 32.2973746212986

r2_score (y_train, y_train_pred)

: 0.472470898145303363

explained_variance_score (y_train, y_train_pred)

: 0.5128831897835

In the above three models of SVR, I think model with Kernel = rbf (Radial **Basis Function Kernel**) is best with 93% accuracy, with best minimum error rate.

Model 3: ***KNN***

Below are parameters used in KNN

n_neighbors: int, optional (default = 5)

Number of neighbors to use by default

for: meth: `kneighbors` queries.

weights: str or callable

weight function used in prediction. Possible values:

'uniform’: uniform weights. All points in each neighborhood are weighted equally.

'distance’: weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

[callable]: a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Uniform weights are used by default.

algorithm: {'auto', 'ball_tree', 'kd_tree', 'brute'}, optional

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use :class:`BallTree`
- 'kd_tree' will use :class:`KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to :meth:`fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size: int, optional (default = 30)

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p: integer, optional (default = 2)

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance

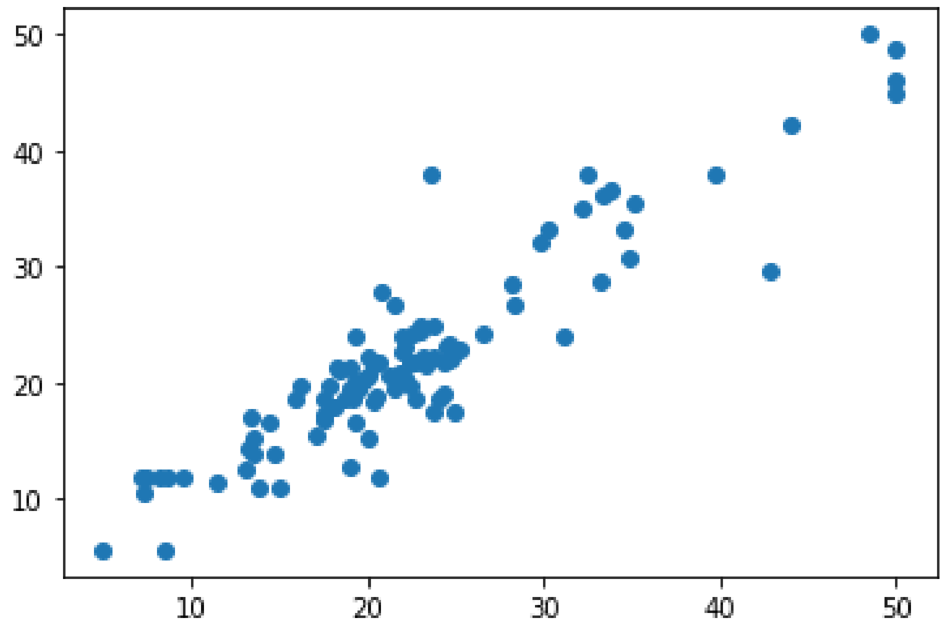
(l2) for p = 2. For arbitrary p, minkowski_distance (l_p)

For this Model I ran grid search for 1 to 100 for n_neighbors, best parameter out of this is n_neighbors = 3

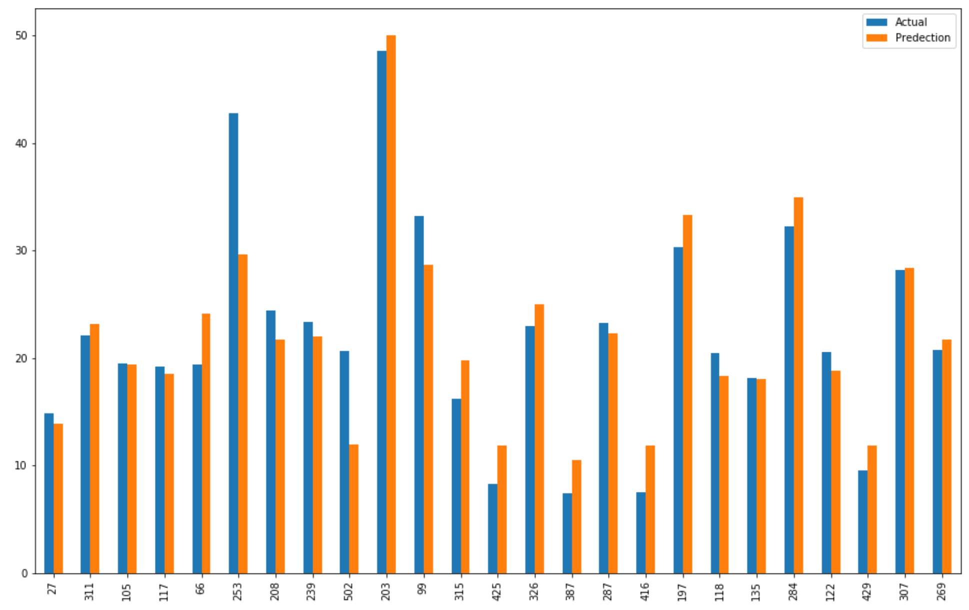
from sklearn.model selection import GridSearchCV

nparams = {'n_neighbors':(range (1 ,100))}

below is the Linear and bar graph



Here we see that we have best fit, with minimum error rate



Prediction vs Actual graph

mean_absolute_error (y_test, y_test_pred)

: 2.369281045751633

mean_squared_error (y_test, y_test_pred)

: 32.2973746212986

r2_score (y_train, y_train_pred)

: 10.863594771241829

explained_variance_score (y_train, y_train_pred)

: 0.8686138404009693

From the Above Three models Linear, SVR, KNN

I would like to choose SVR as the best model for this dataset, compare to all other models, because SVR with Kernel = rbf with gamma = 0.01 has best results compare to all other models with minimum error and prediction is 93%.

References: https://www.aionlinecourse.com/tutorial/machine-learning/support-vector-regression, https://scikit-learn.org , https://www.saedsayad.com.

"This is my own work. I have not copied any of it from anyone else."

Darshan H S