An Introduction to Support Vector Regression (SVR)

Using Support Vector Machines (SVMs) for Regression

Tom Sharp 💻 Mar 4, 2020 · 5 min read ★

Support Vector Machines (SVMs) are well known in classification problems. The use of SVMs in regression is not as well documented, however. These types of models are known as Support Vector Regression (SVR).

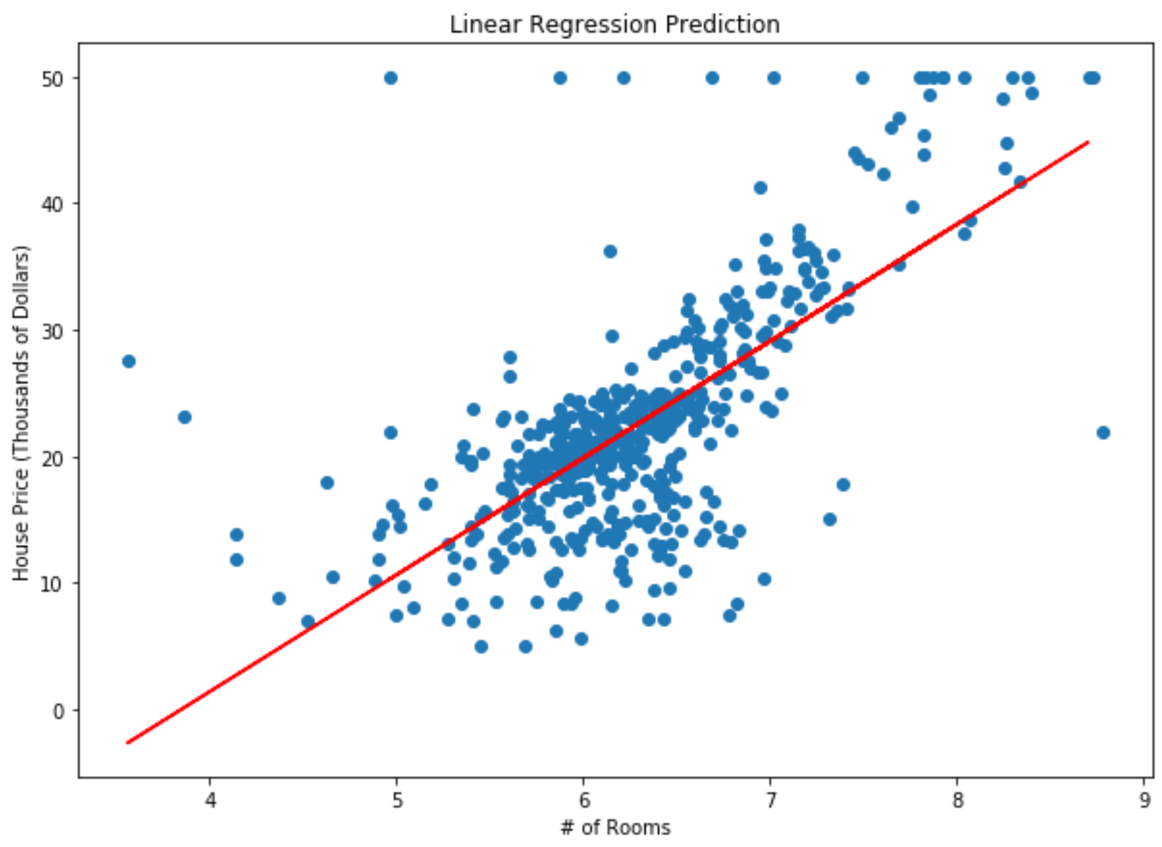
In this article, I will walk through the usefulness of SVR compared to other regression models, do a deep-dive into the math behind the algorithm, and provide an example using the Boston Housing Price dataset.

Simple Linear Regression

In most linear regression models, the objective is to minimize the sum of squared errors. Take Ordinary Least Squares (OLS) for example. The objective function for OLS with one predictor (feature) is as follows:

$$MIN \sum_{i=1}^{n} (y_i - w_i x_i)^2$$

where y_i is the target, w_i is the coefficient, and x_i is the predictor (feature).



OLS Prediction of Boston Housing Prices

Lasso, Ridge, and ElasticNet are all extensions of this simple equation, with an additional penalty parameter that aims to minimize complexity and/or reduce the number of features used in the final model. Regardless, the aim — as with many models — is to reduce the error of the test set.

However, what if we are only concerned about reducing error to a certain degree? What if we don't care how large our errors are, as long as they fall within an acceptable range?

Take housing prices for example. What if we are okay with the prediction being within a

SVR FTW

Enter Support Vector Regression. SVR gives us the flexibility to define how much error is acceptable in our model and will find an appropriate line (or hyperplane in higher dimensions) to fit the data.

In contrast to OLS, the objective function of SVR is to minimize the coefficients — more specifically, the *l*2-norm of the coefficient vector — not the squared error. The error term is instead handled in the constraints, where we set the absolute error less than or equal to a specified margin, called the maximum error, ϵ (epsilon). We can tune epsilon to gain the desired accuracy of our model. Our new objective function and constraints are as follows:

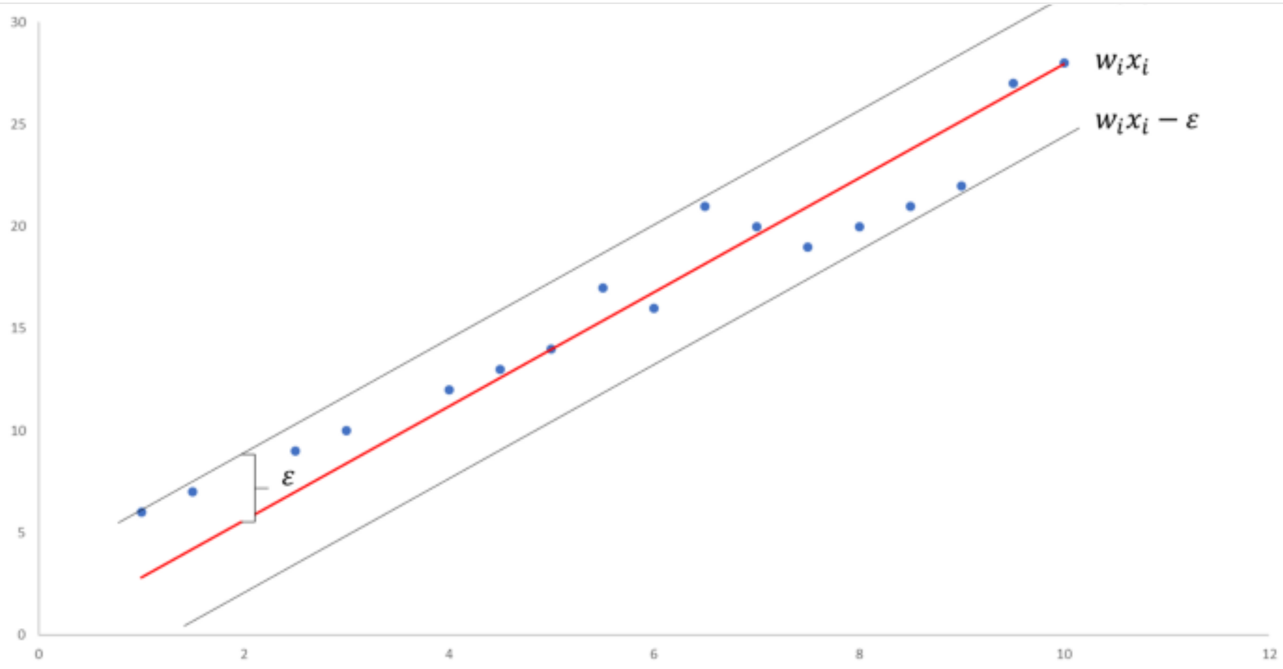
Minimize:

$$MIN \frac{1}{2}||\mathbf{w}||^2$$

Constraints:

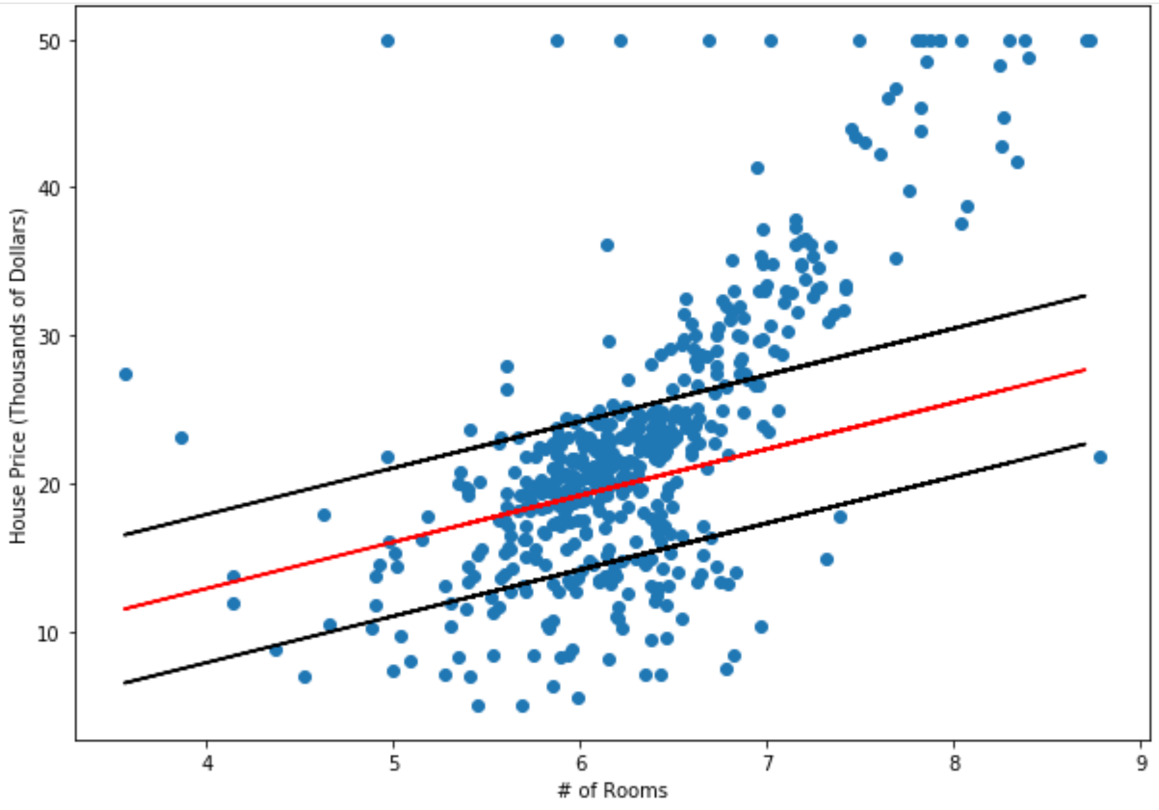
$$|y_i - w_i x_i| \leq \varepsilon$$

Illustrative Example:



Illustrative Example of Simple SVR

Let’s try the simple SVR on our dataset. The plot below shows the results of a trained SVR model on the Boston Housing Prices data. The red line represents the line of best fit and the black lines represent the margin of error, ϵ, which we set to 5 ($5,000).



SVR Prediction of Boston Housing Prices with ϵ=5

You may quickly realize that this algorithm doesn't work for all data points. The algorithm solved the objective function as best as possible but some of the points still fall outside the margins. As such, we need to account for the possibility of errors that are larger than ϵ. We can do this with slack variables.

Giving Ourselves some Slack (and another Hyperparameter)

The concept of slack variables is simple: for any value that falls outside of ϵ, we can denote its deviation from the margin as ξ.

We know that these deviations have the potential to exist, but we would still like to minimize them as much as possible. Thus, we can add these deviations to the objective function.

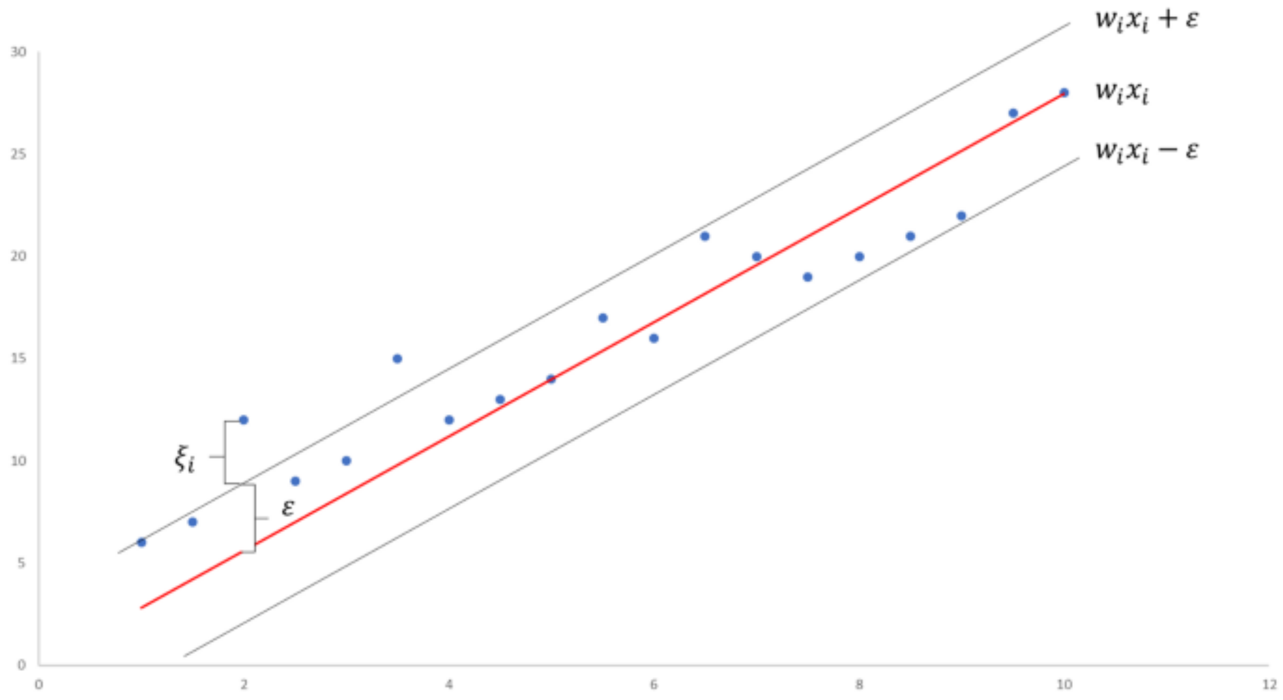
Minimize:

$$MIN\ \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}|\xi_i|$$

Constraints:

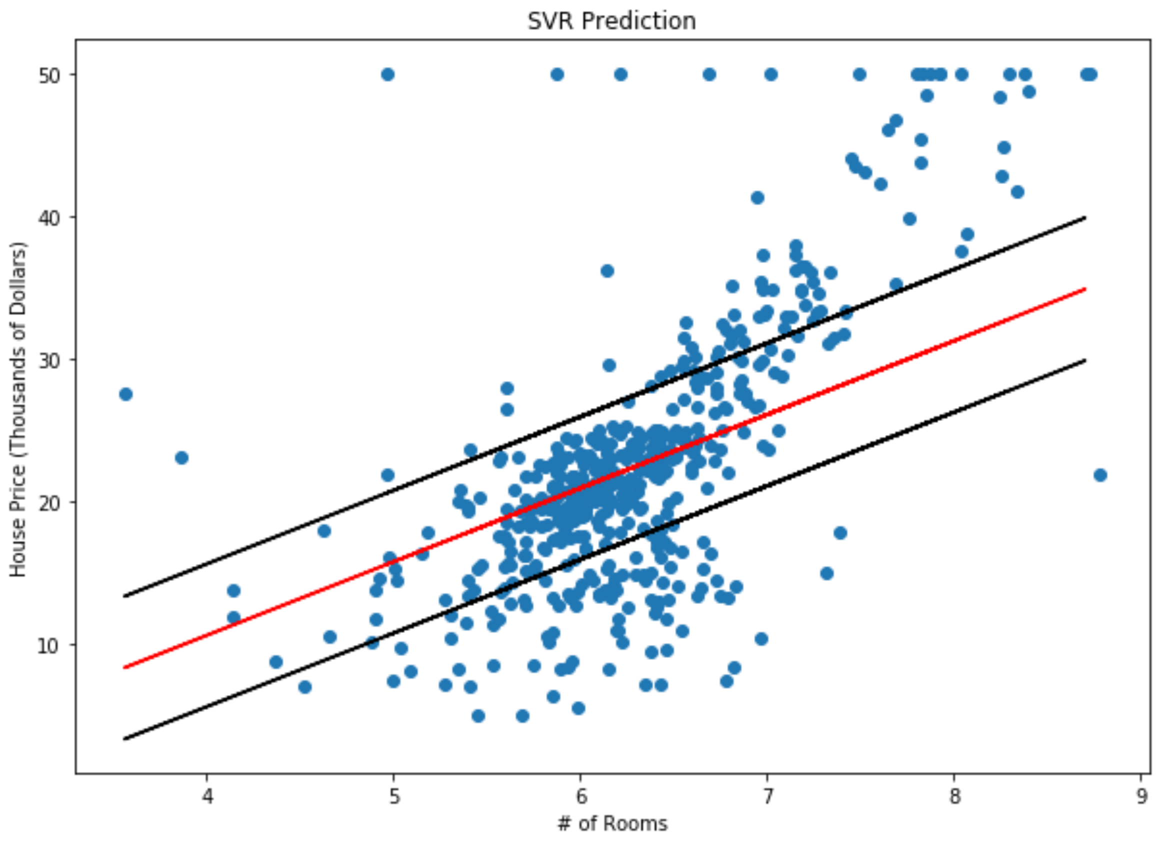
$$|y_i - w_i x_i| \ \leq \ \varepsilon + |\xi_i|$$

Illustrative Example:



Illustrative Example of SVR with Slack Variables

We now have an additional hyperparameter, *C*, that we can tune. As *C* increases, our tolerance for points outside of ϵ also increases. As *C* approaches 0, the tolerance approaches 0 and the equation collapses into the simplified (although sometimes infeasible) one.

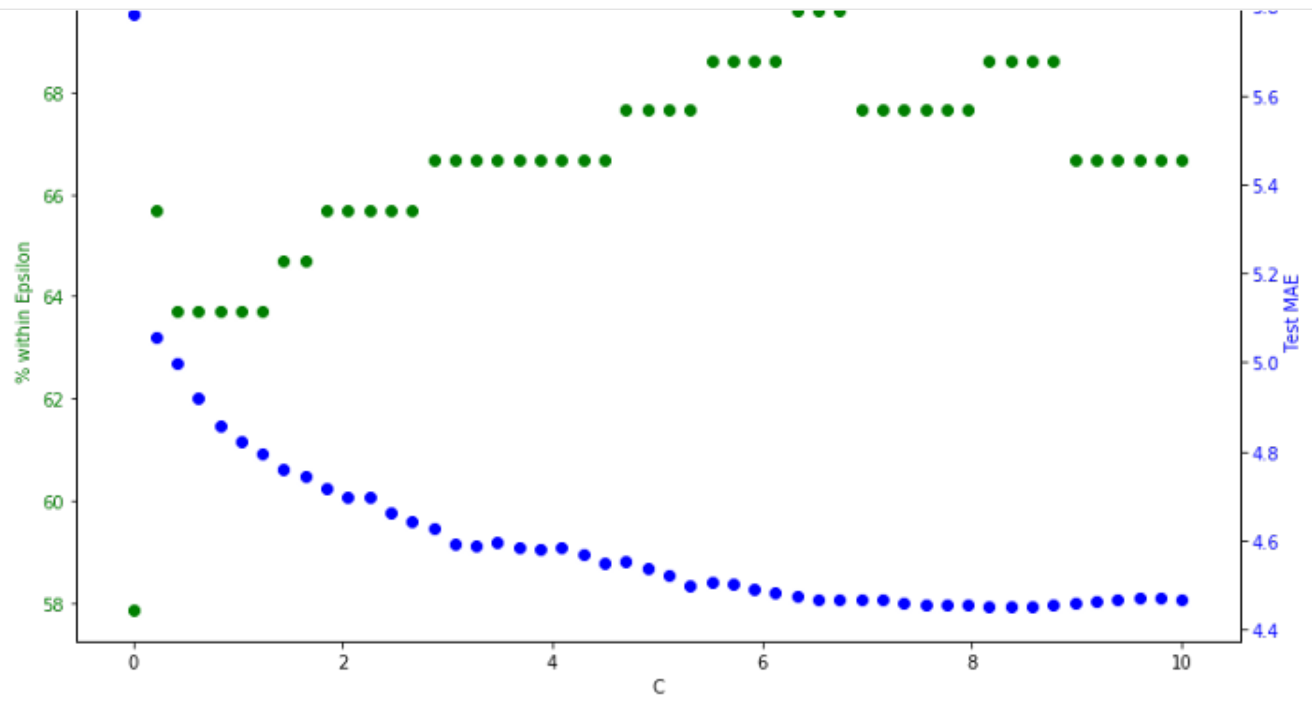


SVR Prediction of Boston Housing Prices with ϵ=5, C=1.0

Finding the Best Value of C

The above model seems to fit the data much better. We can go one step further and grid search over *C* to obtain an even better solution. Let's define a scoring metric, *% within Epsilon*. This metric measures how many of the total points within our test set fall within our margin of error. We can also monitor how the Mean Absolute Error (*MAE*) varies with *C* as well.

Below is a plot of the grid search results, with values of *C* on the x-axis and *% within Epsilon* and *MAE* on the left and right y-axes, respectively.



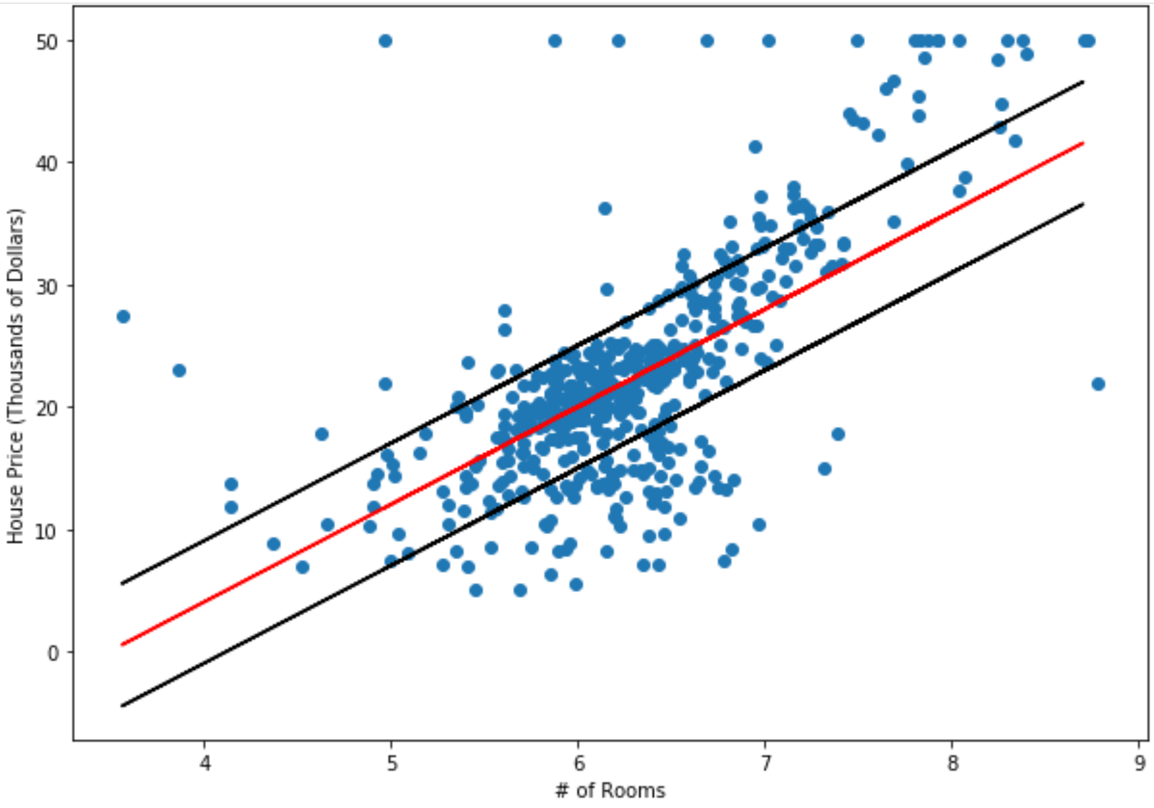
GridSearch for C

The code for this post can be found on my [GitHub](#) page.

As we can see, *MAE* generally decreases as *C* increases. However, we see a maximum occur in the *% within Epsilon* metric. Since our original objective of this model was to maximize the prediction within our margin of error ($5,000), we want t find the value of *C* that maximizes *% within Epsilon*. Thus, *C*=6.13.

Let’s build one last model with our final hyperparameters, ϵ=5, *C*=6.13.

Additional information on this topic: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html https://en.wikipedia.org/wiki/Support-vector_machine#Regression https://www.saedsayad.com/support_vector_machine_reg.htm



SVR Prediction of Boston Housing Prices with ϵ=5, C=6.13

The plot above shows that this model has again improved upon previous ones, as expected.

Conclusion

SVR is a powerful algorithm that allows us to choose how tolerant we are of errors, both throu an acceptable error margin(ϵ) and through tuning our tolerance of falling outside that accept error rate. Hopefully, this tutorial has shown you the ins-and-outs of SVR and has left you confident enough to add it to your modeling arsenal.