Flowforecaster
Traffic flow prediction by using LSR and SVR

<Josh Manto>
<jmm267@duke.edu>

Abstract

This project explores the efficacy of Support Vector Regression (SVR) with various kernels (Linear, Gaussian, RBF, Polynomial) in predicting traffic flow based on time-related features. A novel aspect of our approach is the iterative hyperparameter optimization, particularly for Gaussian and RBF kernels, employing a Bayesian strategy to avoid exhaustive grid searches. Our findings demonstrate significant variances in model performance across different kernels, with optimized parameters yielding substantial improvements in prediction accuracy. This study underscores the importance of kernel selection and hyperparameter tuning in SVR models for traffic flow prediction.

Overview

The project began with preprocessing a traffic flow dataset, followed by splitting it into training and testing sets. I then trained SVR models using four different kernels, applying hyperparameter optimization to enhance model performance. My objective was to identify which kernel size and box constraint parameter set could most accurately predict traffic flow. The project outline is:

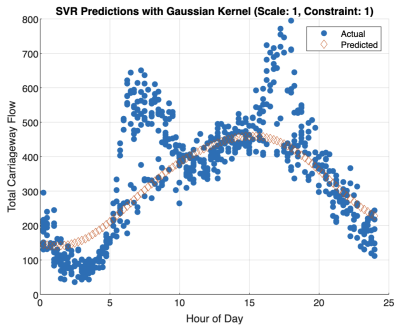
1. Data Preprocessing:
    a. Load the traffic flow dataset.
    b. Split the data into training and testing sets.
2. SVR Model Training and Prediction: (4 separate scripts)
    a. fitrsvm: Use this command to train the SVR models with different kernels.
    b. Gaussian kernel with variations in KernelScale and BoxConstraint.
    c. RBF kernel with variations in KernelScale and BoxConstraint.
    d. Polynomial kernel with different orders by varying PolynomialOrder.
    e. predict: Use this command to make predictions on the test set with the trained SVR models.
3. Hyperparameter Optimization:
    a. Hyperparameter optimization code was appended in each of the four scripts.
4. Joint table of figures and discussion
5. Reflection

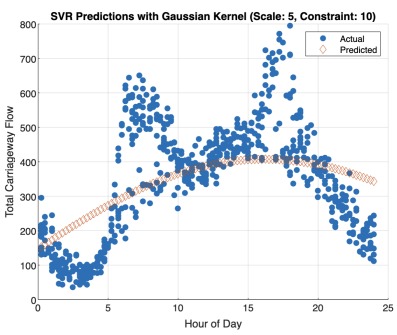
Mathematical Formulation and Implementation

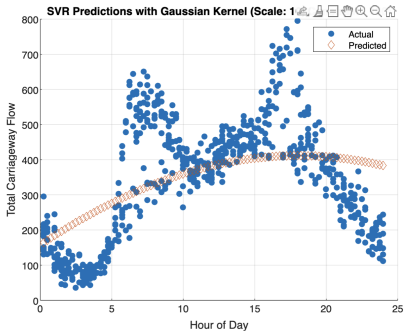
Preprocessing stage (see data_access.m): I focused on predicting total traffic flow using 'LocalTime' and 'LocalDate' as predictors, recognizing that traffic flow significantly varies with the time of day and from day to day. 'LocalDate' was selected to capture potential trends and day-to-day variations within the month. To streamline the analysis, I eliminated all columns except for the chosen predictors and the target variable, 'TotalCarriagewayFlow'. I then divided the dataset into training and testing sets.

Experimental Results

Gaussian Kernel and Hyperparameter Optimization; Figures produced in manual kernel and box constraint adjustments:



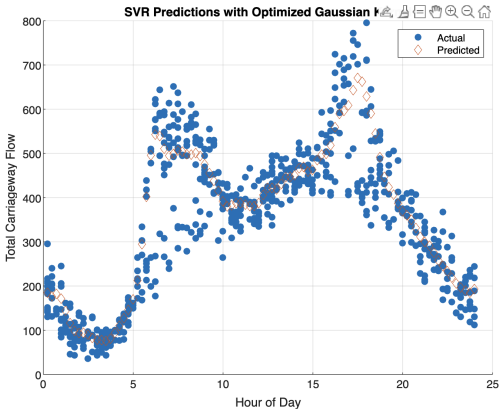




MSE	R2	
10934	0.61923	KERNEL SCALE = auto, BC = 1

MSE	R2	
6484.7	0.77418	KERNEL SCALE = 5, BC = 10

MSE	R2	
10768	0.62501	KERNEL SCALE = 10 BC= 100

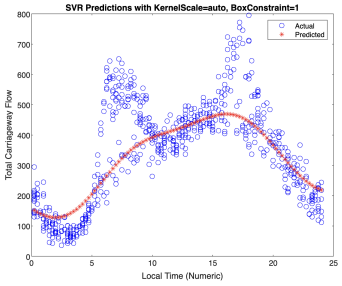


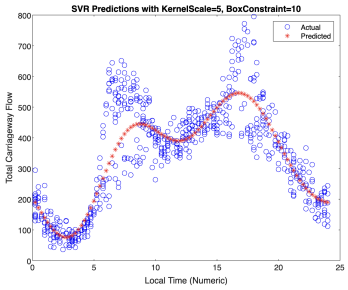
Gaussian hyperparameter optimization:

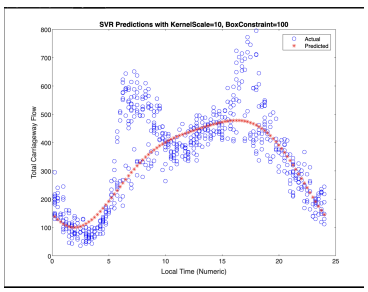
"Gaussian (Scale: 1, Constraint: 1)"	12745	0.55619	""
"Gaussian (Scale: 5, Constraint: 10)"	16968	0.40912	""
"Gaussian (Scale: 10, Constraint: 100)"	18504	0.35561	""
"Gaussian Optimized"	5424.7	0.8111	"KernelScale: 0.027, BoxConstraint: 50.400"

Reflection: In analyzing the performance of the Gaussian kernel for SVR, the results show a clear pattern where a lower KernelScale coupled with a higher BoxConstraint leads to more accurate predictions. Specifically, a KernelScale of 0.027 paired with a BoxConstraint of 50.4 emerged as the most effective combination, achieving an R^2 of 0.8111. This suggests that a finely-tuned balance between the kernel's sensitivity to data points (KernelScale) and the penalty for data points that fall outside the margin (BoxConstraint) is important for model accuracy, and by deviating from these values we decrease the performance of the model.

RBF Kernel and Hyperparameter Optimization; Figures produced in manual kernel and box constraint adjustments:

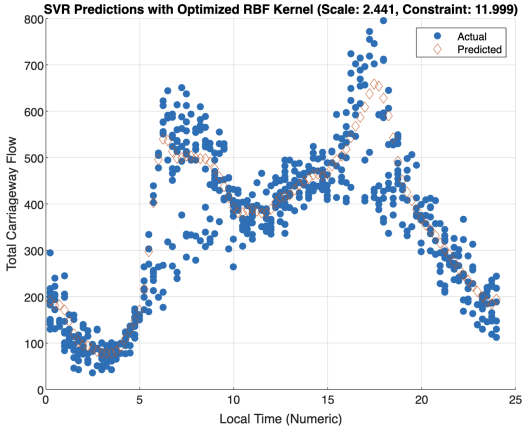






MSE	R2	
10934	0.61923	KERNEL SCALE = auto, BC = 1

MSE	R2	
6484.7	0.77418	KERNEL SCALE = 5, BC = 10



MSE	R2	
10768	0.62501	KERNEL SCALE = 10, BC = 100

RBF hyperparameter optimization:

```
Best Hyperparameters from Optimization:
KernelScale: 2.4409
BoxConstraint: 11.9989
MSE: 5.1860e+03
R2: 0.8194
```

Reflection: For the RBF kernel and hyperparameter optimization, I believe I was successful in applying Bayesian optimization method. I believe RBF and Gaussian approach are quite similar, as they were so fast in calculating the result compared to polynomial. I was happy to also see that at the first try, with KS = 5 and BC = 10, I was able to capture the camel double-hill pattern of the graph. The parameters used in hyperparameter optimization were also close to the values of 5,10, having the best parameters at KS = 2.44 and BC= 12. Surprised that Bayesian approach to look for best parameters work so well with both Gaussian and RBF, I feel relieved that we don't have to do exhaustive grid search.

Polynomial Kernel and Hyperparameter Optimization; Figures produced in manual polyorder changes:

Searching for optimal parameters took a lot of time, because of the time complexity of the operation. I started with an initial search with polynomial order = 8. These are initial results:

```
Polynomial Order: 8, Box Constraint: 5
MSE: 66854844392252039168.0000, R2: -2328104122004482.5000
Polynomial Order: 8, Box Constraint: 10
MSE: 267629582903103913984.0000, R2: -9319736524572120.0000
Polynomial Order: 9, Box Constraint: 1
MSE: 242903363349200051896 32.0000, R2: -845868876971858048.0000
Polynomial Order: 9, Box Constraint: 5
MSE: 20648463664115613126285172439777 28.0000, R2: -719046971187902425812581744 64.0000
```

From project 1, we see that the best fit using the "fit" command was when n=8, but I was initially surprised that this did not fit well. I then understood that Then I modified the search criteria to just iterate between polynomial order 2-5

```
Polynomial Order: 2, Box Constraint: 1
MSE: 12974.8611, R2: 0.5482
Polynomial Order: 2, Box Constraint: 5
MSE: 13416.0237, R2: 0.5328
Polynomial Order: 2, Box Constraint: 10
MSE: 12468.4476, R2: 0.5658
Polynomial Order: 3, Box Constraint: 1
MSE: 419874.6982, R2: -13.6214
Polynomial Order: 3, Box Constraint: 5
MSE: 2297315.4732, R2: -79.0000
Polynomial Order: 3, Box Constraint: 10
MSE: 23170.8276, R2: 0.1931
```

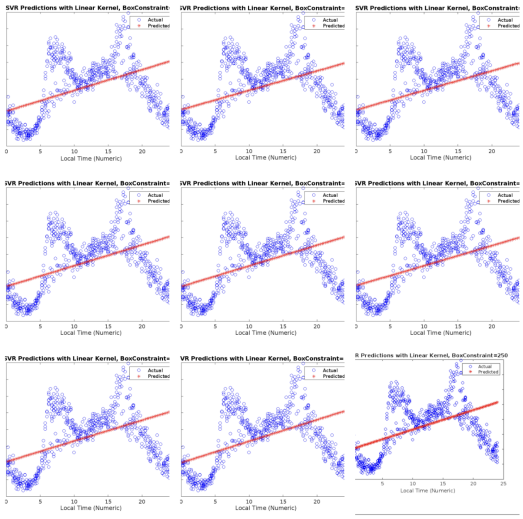
```
Polynomial Order: 4, Box Constraint: 1
MSE: 636502.2046, R2: -21.1651

Polynomial Order: 4, Box Constraint: 5
MSE: 1269939.0594, R2: -43.2234
Polynomial Order: 4, Box Constraint: 10
MSE: 385484.9158, R2: -12.4238
```

It seems to me that polynomial order 2 captures the pattern quite well compared to the rest of the other orders regardless of box constraint settings, the R^2 and MSE seem to just have gradual changes, but it ultimately fails to capture the "camel" or the "two-hill" structure of the data. Polynomial order 3 had given me negative results, this means a cubic function is not best used to capture the pattern. Polynomial order 4 similarly had bad results, and neither do n=8 or 9 yield better results.

Also, I understood that the polynomial kernel also took significantly more time to train compared to Gaussian and RBF perhaps because of the nature of the polynomial computations involved. I also think that perhaps the complexity grows with the number of features in the data. This combinatorial explosion can significantly increase the computational load for each training example as the polyorder is increased.

Linear Kernel and Hyperparameter Optimization; Figures produced in manual kernel and box constraint adjustments:

```
Linear Kernel with BoxConstraint=1
MSE: 25921.0485, R2: 0.0973
Linear Kernel with BoxConstraint=2
MSE: 25922.1001, R2: 0.0973
Linear Kernel with BoxConstraint=3
MSE: 25948.0883, R2: 0.0964
Linear Kernel with BoxConstraint=5
MSE: 25686.7936, R2: 0.1055
Linear Kernel with BoxConstraint=10
MSE: 25651.5084, R2: 0.1067
Linear Kernel with BoxConstraint=15
MSE: 25928.0166, R2: 0.0971
Linear Kernel with BoxConstraint=17
MSE: 25896.5987, R2: 0.0982
Linear Kernel with BoxConstraint=25
MSE: 25763.5441, R2: 0.1028
Linear Kernel with BoxConstraint=50
MSE: 25931.6932, R2: 0.0970
Linear Kernel with BoxConstraint=100
MSE: 26083.0900, R2: 0.0917
Linear Kernel with BoxConstraint=250
MSE: 26182.4771, R2: 0.0882
```



I ran the code iteratively for a set of BC and tested the model accuracy. We can see that the linear kernel performs best when BC = 10, with R^2 = 0.1067.

Final Discussion

I believe after finishing this project, choosing the most appropriate kernel and fine-tuning its hyperparameters is really important. For the Gaussian kernel, a balance between the kernel scale and box constraint was essential for prediction accuracy. A kernel scale of 0.027 and a box constraint of 50.4 yielded the best results with an R^2 of 0.8111. The RBF kernel, on the other hand, favored a mid-range kernel scale and box constraint, successfully capturing the dataset's "camel" pattern. It was observed that a kernel scale of 5 and a box constraint of 10 provided good fit, with optimization through Bayesian methods enhancing performance further to an R^2 of 0.8194.

The polynomial kernel was the hardest challenge due to its computational intensity, especially at higher orders which resulted in overfitting and negative R-squared values. The outcomes I got suggest that higher polynomial orders in SVR models behave differently compared to least squares regression, perhaps due to the SVR's regularization aspect that penalizes overfitting. Of course, least squares yielded similar results showing upward trend regardless of using fitsrvm command or fit command, but I would still use least squares.

For future improvements, I will continue using Bayesian optimization as it a very time-efficient strategy that eliminates exhaustive grid search typically associated with hyperparameter tuning. It would also be advisable to use kernels that do not have high time-complexity, such as Gaussian or RBF, as polynomial curve fitting had the longest training time, and the worse results of them all.

Some additional implementations I did was implementing a grid-search and random-search algorithm to look for best hyperparameters in Gaussian kernel and compared the training time; it yielded the following results:

Random Search Results:

"Gaussian (Random Scale: 1.00, Constraint: 10.00)"	9327.3	0.67519	""
"Gaussian (Random Scale: 5.00, Constraint: 1.00)"	25139	0.12456	""
"Gaussian (Random Scale: 0.50, Constraint: 50.00)"	5239.3	0.81755	""
"Gaussian (Random Scale: 1.00, Constraint: 10.00)"	9327.3	0.67519	""
"Gaussian (Random Scale: 0.50, Constraint: 10.00)"	5223.5	0.8181	""
"Gaussian (Random Scale: 0.50, Constraint: 50.00)"	5239.3	0.81755	""
"Gaussian (Random Scale: 2.00, Constraint: 10.00)"	12970	0.54833	""
"Gaussian (Random Scale: 2.00, Constraint: 10.00)"	12970	0.54833	""
"Gaussian (Random Scale: 0.10, Constraint: 10.00)"	5423.3	0.81114	""
"Gaussian (Random Scale: 2.00, Constraint: 50.00)"	12883	0.55138	""

Random Search Training Time: 1.3436 seconds

Grid Search Results:

Kernel	MSE	R2	Details
"Gaussian (Scale: 0.10, Constraint: 0.10)"	26619	0.073023	""
"Gaussian (Scale: 0.10, Constraint: 1.00)"	11662	0.59388	""
"Gaussian (Scale: 0.10, Constraint: 10.00)"	5423.3	0.81114	""
"Gaussian (Scale: 0.10, Constraint: 50.00)"	5787.9	0.79845	""
"Gaussian (Scale: 0.10, Constraint: 100.00)"	5836.5	0.79675	""
"Gaussian (Scale: 0.50, Constraint: 0.10)"	21509	0.25099	""
"Gaussian (Scale: 0.50, Constraint: 1.00)"	8097	0.71804	""
"Gaussian (Scale: 0.50, Constraint: 10.00)"	5223.5	0.8181	""
"Gaussian (Scale: 0.50, Constraint: 50.00)"	5239.3	0.81755	""

```
    "Gaussian (Scale: 0.50, Constraint: 100.00)"    5282.9       0.81603      ""
    "Gaussian (Scale: 1.00, Constraint: 0.10)"       21774       0.24175      ""
    "Gaussian (Scale: 1.00, Constraint: 1.00)"       12745       0.55619      ""
    "Gaussian (Scale: 1.00, Constraint: 10.00)"     9327.3       0.67519      ""
    "Gaussian (Scale: 1.00, Constraint: 50.00)"     7156.8       0.75078      ""
    "Gaussian (Scale: 1.00, Constraint: 100.00)"    6550.2        0.7719      ""
    "Gaussian (Scale: 2.00, Constraint: 0.10)"       25363       0.11679      ""
    "Gaussian (Scale: 2.00, Constraint: 1.00)"       14662       0.48942      ""
    "Gaussian (Scale: 2.00, Constraint: 10.00)"      12970       0.54833      ""
    "Gaussian (Scale: 2.00, Constraint: 50.00)"      12883       0.55138      ""
    "Gaussian (Scale: 2.00, Constraint: 100.00)"     12746       0.55614      ""
    "Gaussian (Scale: 5.00, Constraint: 0.10)"       28849    -0.0046313      ""
    "Gaussian (Scale: 5.00, Constraint: 1.00)"       25139       0.12456      ""
    "Gaussian (Scale: 5.00, Constraint: 10.00)"      16968       0.40912      ""
    "Gaussian (Scale: 5.00, Constraint: 50.00)"      13042       0.54585      ""
    "Gaussian (Scale: 5.00, Constraint: 100.00)"     12858       0.55223      ""

Grid Search Training Time: 3.5882 seconds
```

Reflection: The Random Search method, notably quicker with a training time of 1.3436 seconds, identified a combination yielding an R^2 of 0.8181 with a KernelScale of 0.50 and a BoxConstraint of 10.00. This demonstrates the efficacy of Random Search. Grid Search, while more systematic and covering a broader range of combinations, took longer, at 3.5882 seconds. However, it provided a comprehensive overview of how different combinations of KernelScale and BoxConstraint impact the model's performance. The most notable result from Grid Search was an R^2 of 0.8181 with a KernelScale of 0.50 and a BoxConstraint of 10.00. Both were able to converge into good solutions quickly. Random search can therefore be as effective as grid search in identifying hyperparameters. I suppose the same would be the case when doing RBF.

Combined kernel predictions: