Bike Rental Prediction for Capital Bike Share

Prasanna K. Challa **Sagar Ghiya** **Yizhen Chen**

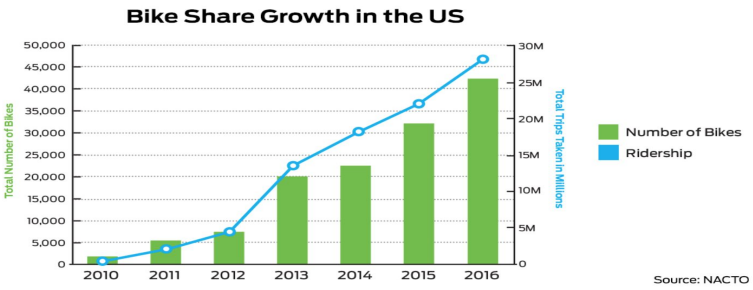
Abstract

The goal of this project is to predict the number of bikes rented in a given hour of the day for the Capital Bike Share company located in Washington D.C. The idea is to implement different supervised machine learning algorithms to predict the count of bikes rented using time and weather factors as predictor variables. Models were compared using RMSE (Root Mean Squared Error) as an evaluation metric and the model with least RMSE is selected for future predictions.

1. Introduction

Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. Through these systems user can easily rent a bike from a particular position and return back at another position. Currently, there are about 500 bike sharing programs around the world which is composed of over 500 thousand bicycles. Today there exists great interest in these systems due to their important role in traffic, environmental and health issues.

Apart from interesting real world applications of bike sharing systems, the characteristics of data being generated by these systems make them attractive for the research. Opposed to other transport services such as bus or subway, the duration of travel, departure and arrival position is explicitly recorded in these systems. This feature turns bike sharing system into a virtual sensor network that can be used for sensing mobility in the city.



Moreover, due to high variation in the demand of bikes from time to time it becomes very important for the company to keep track of the demand for the bikes that will be rented each hour to efficiently allocate their resources and optimizing their budget. This will ensure an uninterrupted access to a faster and easier way for people to reach their destination, in cases when public transportation is either not available or inconvenient, and when people who don't have a car or when parking is very limited at that destination and many other reasons. By using features such as weather, season, holiday, working day we aim to predict the variable "cnt" with as much accuracy as possible.

2. Technical Approach

Four machine learning models were implemented in this project – Ridge Regression, Neural Networks, Support Vector Regression and Random Forests. The models were tuned and compared using RMSE as the evaluation metric.

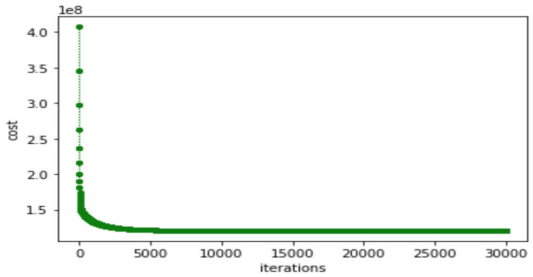
2.1 Ridge Regression

Ridge Regression is used as a baseline model to use for performance comparison with other complex models. We tried simple linear regression model without adding any regularization term initially which resulted in some extent of overfitting on train data. L2 regularization was then included in the cost function to address the problem of high variance, which resulted in better generalization on validation set. The regularization parameter α in below equation can be used to control the amount of regularization in the model (overfitting decreases as α increases), thereby helping in maintaining a good bias-variance tradeoff.

$$\ell(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta x^{(i)} - y^{(i)})^2 + \alpha\|\theta\|^2$$

The plot below shows that the cost function reaches a global optimum and the parameter update got slowed down in first 4000 iterations as the model is relatively simple

Cost as a function of number of iterations:



The performance of the implemented custom model (Test RMSE: 187.274) was finally compared with the model from sklearn library in python (Test RMSE: 187.278) and observed to be performing almost similar, but produced slightly better results than the latter.

2.2 Neural Networks

A deep neural network with 4 hidden layers (20, 25, 25, 20) was designed to try and exceed the performance of ridge regression model. The cost function we employed is least squares error with L2 regularization. The input layer has 12 nodes as we have 12 input features and the output layer has 1 node. ‘ReLU’ was chosen to be the activation function for all hidden units as it generally outperforms both ‘sigmoid’ and ‘tanh’ activation in most of the cases, and effectively addresses the problem of vanishing gradients at initial layers. We used ‘ReLU’ at output layer as well, as we are dealing with a regression problem. The activations from the output unit were

rounded off to closest integers as the target variable ‘count’ that we are predicting consists only values of type integer.

$$J_\lambda(w,b) = \frac{1}{m}\sum_{i=1}^{m} J(w,b,x^{(i)},y^{(i)}) + \frac{\lambda}{2}\sum_{l=1}^{n-1} \| w^{(l)} \|_F^2$$

$$J(w,b,x^{(i)},y^{(i)}) \triangleq \frac{1}{2} \| h_{w,b}(x^{(i)}) - y^{(i)} \|_2^2$$

To implement the neural network using TensorFlow we employed ‘Gradient Descent’ as the optimizer using mini batch size of 128 and regularization parameter $\lambda = 5$. Decreasing the batch size is significantly increasing the computational time but works effectively in reducing the error

We observed that when high learning rate (0.001) is used, the cost function is taking large steps to find the optimal solution but it is overshooting the minimum and not finding optimal solution. But this will speed up the algorithm. And when the learning rate is low (0.0001), the algorithm is more reliable in finding the global minimum but takes significant amount of time to converge as steps to reach the minimum of loss function are very tiny.

To address this problem, we created a new hyperparameter ‘percent’, which indicates the percent of epochs or iterations after which a new learning rate should be used for the optimization. This new hyperparameter can be used to specify 2 different learning rates for the model. We use a higher learning rate (0.001) for first n% of iterations where it would be beneficial to take bigger steps to reach the global minima and thereby reducing computational time, and after the specified n% iterations are completed we use a smaller learning rate (0.0001) as its better to take smaller steps at this level to avoid overshooting of the gradient.

This technique has substantially improved the computational efficiency of the algorithm by reaching the global optimum in much lesser amount of time, and reduced the error rate significantly.

The custom model performance (Test RMSE: 119.138) is far better compared to the model in sklearn library (Test RMSE: 240.857).

2.3 Support Vector Regression

Support vector regression with linear kernel was implemented using the modified SMO algorithm for regression. The algorithm aims to find the optimal values of lagrangian multipliers ensuring the KKT conditions are not violated. There were 4 main hyperparameters- C, Epsilon, tolerance and number of iterations, that we mainly focused on to optimize the performance of the model.

The hyperparameter C in the model determines the trade off between the model complexity (flatness) and the degree to which deviations larger than epsilon are tolerated in optimization

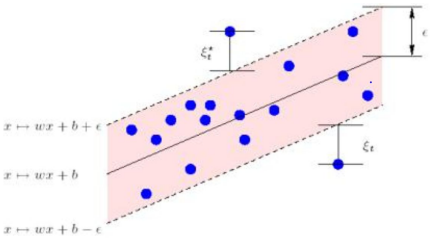
formulation. We checked the condition when C is set to very large value (C=500) and noticed a very high error rate, which is due to the fact that the objective function aims to minimize the empirical risk only in this case, without regard to model complexity in the optimization formulation. Model was using different values of C, and was observed that increasing the value of C until certain threshold (C=1) has improved the performance of the model considerably by allowing model to make some limited amount of mistakes.

Epsilon in the model was used control the width of the insensitive zone, used to fit the training data. Epsilon affected the number of support vectors used to construct the regression function. Bigger the value of epsilon, fewer the support vectors that were selected. On the other hand, bigger values resulted in more 'flat' estimates.

The RMSE on validation set decreased by decreasing the value of 'tolerance' until a certain threshold. We increased the value of number of iterations starting with 100 till the algorithm converged and reached the global optimum.

$$\min \quad \frac{1}{2}\| w \|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

$$\begin{cases} y_i - f(\mathbf{x}_i, w) \le \varepsilon + \xi_i^* \\ f(\mathbf{x}_i, w) - y_i \le \varepsilon + \xi_i \\ \xi_i, \xi_i^* \ge 0, i = 1, \ldots, n \end{cases}$$



2.4 Random Forests

The Random Forest model was designed using the sklearn library in python. We initially made N bootstraps from both the training and testing data and each bootstrap is a randomly chosen subset of the whole training dataset. After that, we build the decision tree for each training bootstrap and form a forest. Each decision tree is regarded as a specialist and will return the best result based on it's knowledge. Also, any two decision trees should be independent. So it provides this algorithm with good generalization ability and avoids overfitting. For testing part, we put each testing bootstrap in the forest and got the output generated by each decision tree in the forest and finally used the average of all outputs as the result.

In this algorithm, there are 4 hyperparameters that significantly affected the regression accuracy. The N_estimators: When the number of tree is too small in the forest, the algorithm will have high risk of overfitting. With the increment of this hyperparameter, the algorithm will become more and more stable, but more time consuming. The Max_features: By the time we split the node in a decision tree, we randomly select a subset of feature and choose one feature that can best divide the data into two parts.This hyperparameter restricts the number of randomly chosen feature when making split on each node.And to make the algorithm more accurately,this hyperparameter should less than the number of whole features and even far less than if possible.The MAX_Depth:If the maximum length is too small, we will lose many important information and will increase the error significantly. The judge function is used to measure the

quality of each split. Different judging functions will result in different accuracy on the algorithm, but just affect in a small range.

When we form each bootstrap, the rest of the data is called Out Of Bag (OOB) data from which we calculate the error for a certain feature x of the data and called OOB1.Then we add some random noise into the same OOB data and do the same error calculation on the same feature and call this OOB2. Importance = ∑(errOOB2-errOOB1)/N.

If the error for a certain feature changes dramatically after adding random noise,it means this factor takes an important role in the regression and the larger it changes, the more important it would be.

3. Experimental Results

The data is collected from UCI Machine Learning Repository and has 17389 instances with 17 features.It contains the hourly count for bikes rented between years 2011 and 2012 in Washington, DC. The variables such as instant (indicating the record index as row numbers), dteday (date),casual and registered(indicating the number of casual and registered users) which sum up to give the target variable count are excluded, as they are not appropriate to include in modeling. All the other numeric variables are normalized using min-max normalization as (t - t_min)/(t_max - t_min). The target variable 'cnt' is of type integer with minimum=1, maximum=977, mean=189.4 and has very high variance.

3.1 Feature Information:

Time Attributes	Values
season	1:Spring, 2:Summer, 3:Fall, 4:Winter
yr	Year (0: 2011, 1:2012)
month	Month (1 to 12)
hr	Hour (0 to 23)
holiday	Government holiday (0:no 1:yes)
weekday	Weekdays (0 to 6)
working day	Working day (0:no 1:yes)

Weather Attributes	Values
weathersit	1:Clear, 2:Cloudy, 3:Light snow or rain, 4:Heavy snow or rain
temp	Temperature numeric: [0,1]
atemp	Feeling temperature numeric: [0,1]
hum	Humidity numeric: [0,1]
wind speed	Wind Speed numeric: [0,1]

3.2 Splitting the data and hyperparameter tuning:

For each algorithm, data is divided as 60% for training, 20% for validation and 20% for testing.
The models were trained using train set, tuned using validation set, and final results were generated using the test set. We used for loops to tune the SVM model by tuning one hyperparameter at a time by keeping all the other hyperparameter fixed initially until we got decent set of hyperparameter values. And later tuned by focusing on the associations betweens different hyperparameters to find the best combination of hyperparameters.

Optimized values of hyperparameters for each algorithm:

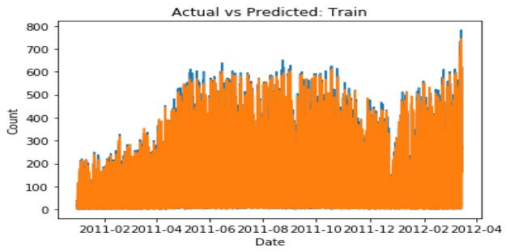
Algorithm	Optimized hyperparameters
Ridge regression	alpha=0.0000004,lmda=0.2,iterations=30000,intercept=True
Neural Network	hidden_layer_sizes=(20,20,20,20),activation='relu',alpha=0.1,batch_size=128, learning_rate_init=0.0001, max_iter=500
SVM regression	C = 1, epsilon = 0.0001, tol = 0.00001, kernel_type = linear
Random Forest	n_estimators=500,criterion="mse",max_features=12, max_depth=auto

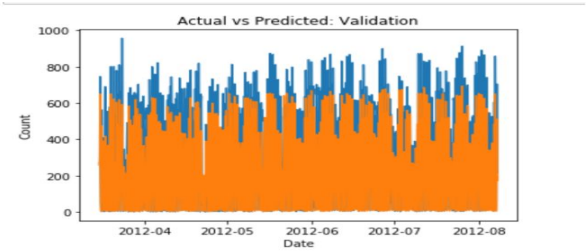
Comparison of performances of all the algorithms by RMSE values:

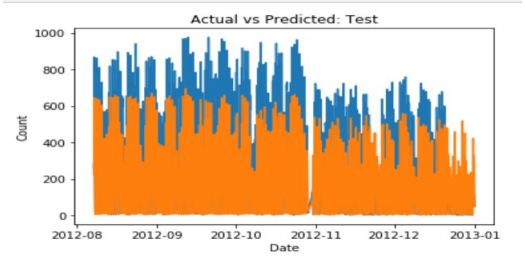
Algorithm name	RMSE in training	RMSE in validation	RMSE in testing
Ridge regression	107.4178	184.7357	187.2781
Neural Network	64.9958	106.2340	119.138
SVM Regression	109.7181	198.2046	197.6296
Random Forest	12.7187	100.0399	113.5584

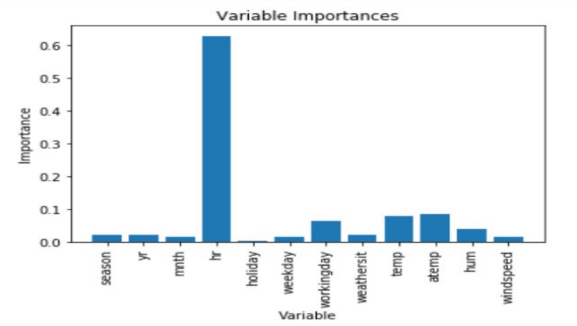
The Random Forest algorithm has the smallest RMSE error for all training,validation and testing followed by Neural Network. Hence, random forest model is the best model to be used by the company for future predictions of bike share count.

The first 3 plots below shows the predicted values (‘orange’) and true values (‘blue’) of the target variable count on y-axis plotted against the ‘date’ variable on x-axis for random forest. The fourth plot shows variable importance generated by random forest model.The variable importance plot shows that ‘Hour’ followed by ‘atemp’.









4. Participants Contribution

Prasanna K. Challa: Prasanna implemented the Ridge Regression and Neural Networks algorithms in the project which gave better results than the sklearn models and also performed data pre-processing. He has a sound knowledge of all the algorithms and their implementations, so he also helped others with debugging code or any other issues in the group to accelerate the project.

Sagar Ghiya: Sagar implemented Support Vector Regression with SMO for the project along with tuning hyperparameters to improve accuracy. He also did the project proposal presentation in the classroom.

Yizhen Chen: Yizhen implemented Random Forest algorithm in the project. He successfully tuned the parameters in the algorithm which ultimately gave the best accuracy. He also presented the final project presentation in the classroom.

Apart from the individual work, all 3 of us put in combined efforts for executing remaining work including sharing ideas, preparing presentations and reports for the project.

References

[1] Fanaee-T, Hadi, and Gama, Joao, 'Event labeling combining ensemble detectors and background knowledge', Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg

[2] http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

[3] Adele Cutler,'Random Forests for Regression and Classification',Utah State University, page 78-page 100

[4] Segal,Mark R,'Machine Learning Benchmarks and Random Forest Regression':UC San Francisco,2004-04-14

[5] Improvements to SMO Algorithm for SVM Regression