Bike Sharing Demand

Kaggle Challenge 2015

Piyush Kumar | Prashant Kumar | Shujaat Ishaq | Swaroop Dewal

COURSE PROJECT CS771, IIT KANPUR

HTTPS://GITHUB.COM/PIYUSHPANKAJ/KAGGLE-BIKE-SHARING

This project was done under the supervision of Dr. Harish Karnick for the course Machine Learning Techniques in spring semester 2015.

Contents

1. Introduction

Bike renting is the new form of public transportation where people can rent bikes at one location and return it back to different location. Bike sharing systems facilitates the above. It mainly consists of two components namely the IT Systems and renting stations. IT systems are responsible for maintaining user registration data, tracking demand at stations and redistributing this demand to different stations. Bike sharing systems automates this via a network of kiosk locations throughout a city. Renting stations are kiosk locations from where bikes are retrieved or released as per the demand. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city.

Figure 1.1: Bike renting kiosk locations

1.1 Motivation

Prediction of sharing demand of transportation systems like cab services has gained a lot of attention in machine learning community. The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Increased traffic in cities and environment friendly nature of bikes has made this mode of transportation as a viable options for daily commuters.

1.2 Problem Statement

Based on the historical usage pattern of bike rentals along with weather conditions we need to forecast the bike rental demand in the Capital Bikeshare program in Washington, D.C. In this Kaggle challenge we are provided with the hourly bike rental data for the first 20 days of each month for the years 2011 and 2012. Our job is to predict total number of bikes rented on an hourly basis for the last 10 days of every month.

1.3 Dataset

The dataset used in this project is available at Kaggle and is collected from Capital Bikeshare program in two years(2011 and 2012). The trainset contains 10886 number of entries which are hourly count of bike rentals for first 20 days of every month. The test data contains 6493 entries same which are hourly count of bike rentals for last 10 days of every month. Both trainset and testset has 9 features in common and trainset has three additional outcome variables namely casual,registered and count. Following are the details of the dataset.

Dataset Information	
Feature/Labels	ValueType
Date-TIme	DD-MM-YY HH:MM:SS
Season	1 = spring, 2 = summer, 3 = fall, 4 = winter
Holiday	whether the day is considered a holiday
WorkingDay	whether the day is neither a weekend nor holiday
Weather	• 1: Clear, Few clouds, Partly cloudy, Partly cloudy • 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist • 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds • 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
Temp	temperature in Celsius
Atemp	"feels like" temperature in Celsius
Humidity	relative humidity
windspeed	wind speed
Casual	number of non-registered user rentals initiated
Registered	number of registered user rentals initiated
Count	number of total user rentals initiated

1.4 Evaluation Criteria

Kaggle uses root mean logarithmic squared error(RMLSE) to calculate accuracy of the model. This is defined as follows:

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

Figure 1.2: Evaluation Criteria

where:

- **n** is the number of samples in the test set
- $\mathbf{p_i}$ is the predicted number of bikes rented
- $\mathbf{a_i}$ is the actual number of bikes rented

2. Understanding your data

2.1 Pre-Processing Raw Data

In the last section, we saw the Dataset available to us and the features present in the training and test data. But before feeding this raw data to the model we first need to some do some pre-processing.

- **Remove non-numerical data** : To ensure better functioning pf our linear and non-linear model some non-numeric features are transformed into separate numeric features. We parsed **datetime** into separate variables like **year**, **month**, **day** and **hour**.
- **Transforming the outcome variables** : Regression models generally minimizes Root-Mean square errors (RMSE) while training. But as discussed above, this challenge evaluates on the basis of Root Mean Logarithmic Squared error(RMLSE). So to ensure that RMLSE is minimized, we scaled up our outcome variables(count, registered and casual) to their logarithmic values. After prediction this value is scaled down to ensure compatibility.

2.2 Deciding What To Do

Before doing any feature engineering we first need to know about our data. Here are few steps that tells us few things about our predictors and their correlation between each other and their relation with the outcome variables.

2.2.1 Feature Importance

We first tried to plot feature importance to get some insight about the importance of features with the total number of bike rented. As it is clear from the following plot that bike rented heavily depends on hour and there is very less dependency on the fact that whether it's a holiday or not [1].

[1]For example purposes this feature importance is plotted for random forest model, but WLOG we can say that this holds for almost all the models

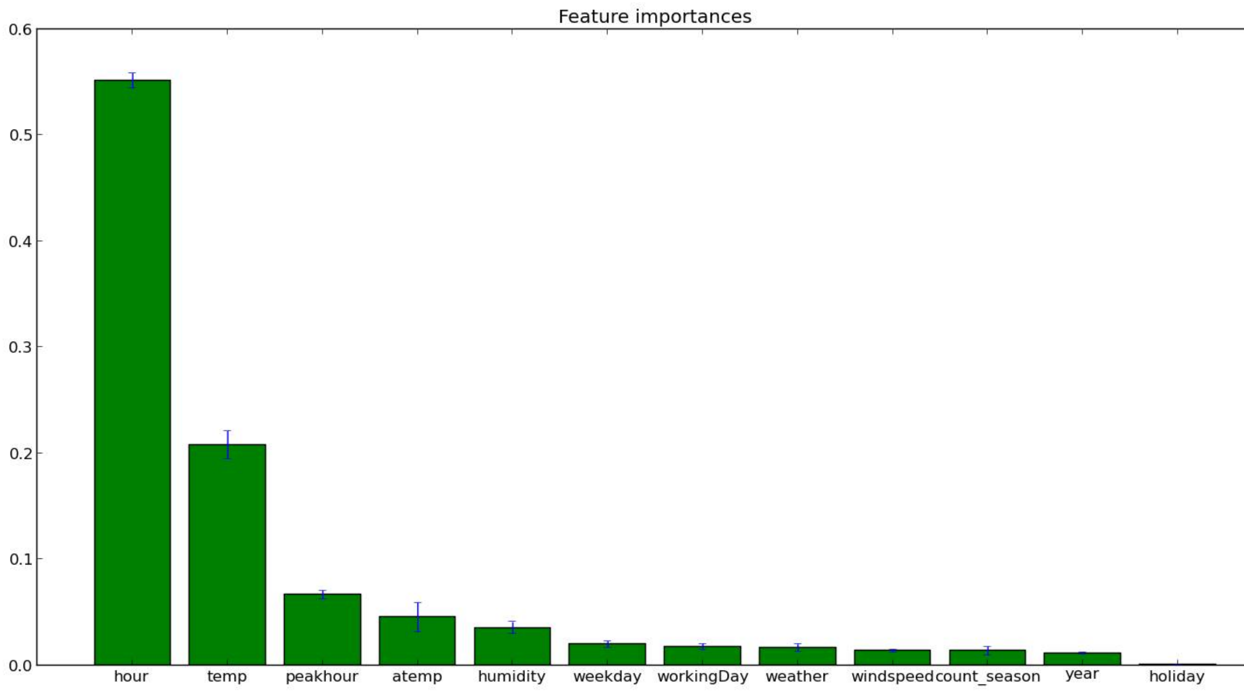


Figure 2.1: Feature Importance

2.2.2 Correlation between Hour and count of Bike Rented

Due to the high dependency of **hour** variable with the number of bike rented. It's essential to dive deep into this and find out how are they related to each other.

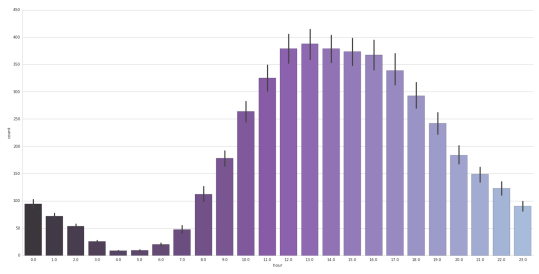


Figure 2.2: On Weekdays

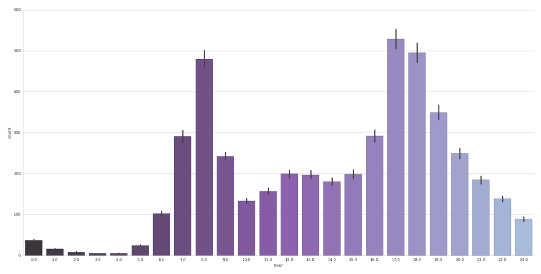


Figure 2.3: On Weekends

As we can see that on weekdays there are two intervals where bike rental demands were at peak. Same is the case with weekends where bike rentals demands were at peak between 10pm to 7 pm.

2.2.3 Correlation between Month and number of Bikes rented

From the above plot it can be inferred that all the essence of **month** is captured by the season variable. Also month was not so important feature, so removing this will not have much effect on aaccuracy. On the other hand it will reduce the complexity of the model by a large factor.

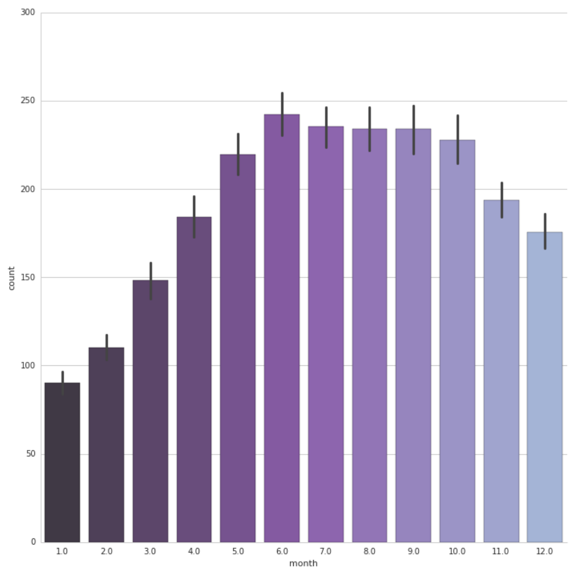


Figure 2.4: Month vs Bikes Rented

2.3 Feature Engineering

Before feeding any raw data to model, we want to increase the correlation between the features and the outcome variable(**i.e** Number of bikes rented). Here are few state-of-art ways to engineer raw data.

2.3.1 Digitization

Some variables are categorical in nature, for example we have season and weather as categorical variables with 4 categories. This category is created assuming that they are independent to each other. But in many cases this doesn't holds true. One way is to binarize these categorical variables into separate variables like **spring, summer, autumn and winter** in case of season variable. This digitization can be applied to some other categorical variables as well like weather, month, hour, weekday etc.

2.3.2 Conditional Expectation Value Mapping

We can transform categorical varibale into their expected value of total number of bikes rented. This can exploit the ordering between different categories in case of categorical variables. We used this approcah for season and weather variables.

2.3.3 Removing redundancies from the Dataset

In this dataset we can see that their is high correlation between **atemp** and **temp** variables ($\tilde{9}$8-99%). We can safely ignore of them as it will make our model less redundant. Also we can see that there is also two instances of **weather = 4** in train data and one instances in test data. To handle this sparseness of data we transformed all these instances to **weather = 3**.

2.3.4 Introducing a new variable Peak Hour

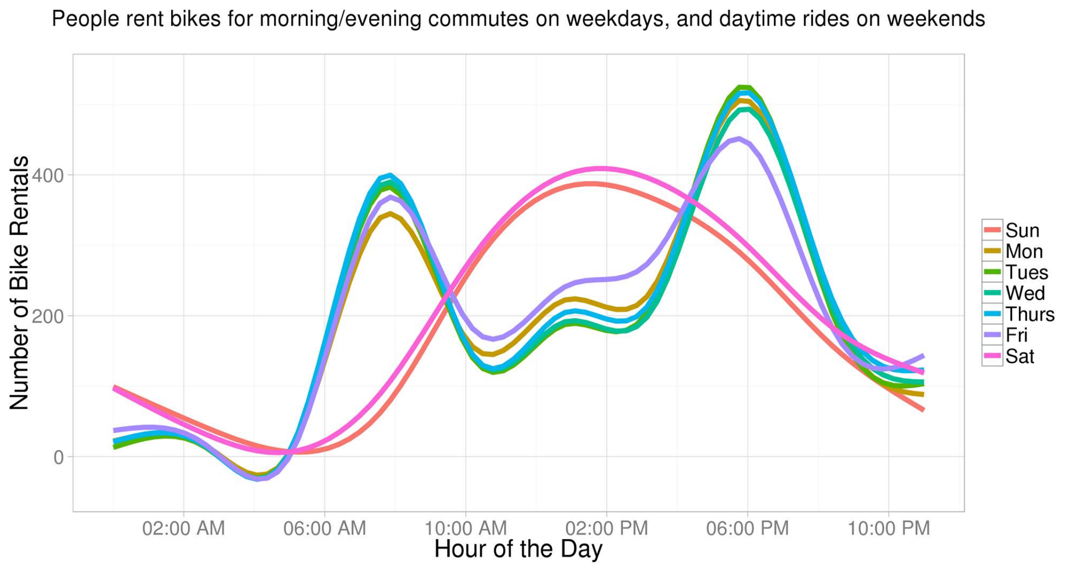
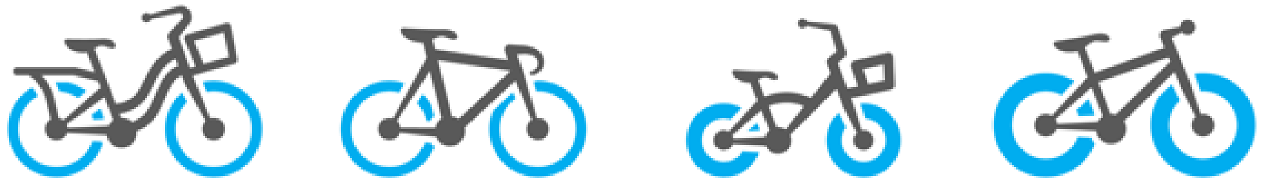


Figure 2.5: Peak Hour

As discussed in the above section, we introduced a new variable peak hour. This captures the those hours when there is maximum demand of bikes.



3. Model Selection

We tried to fit different models on our data. We will be discussing models in decreasing order of the testing error or in increasing order of the model performance.

3.1 Linear Models

We experimented with three linear models namely ridge regression, lasso regression and elastic net. Each of them are explained below.

3.1.1 Ridge Regression

Ridge regression is a modified version of Linear regression with an added penalty(regularization) term which accounts for model complexity. It minimizes the squared loss function with a regularization term which is the L2 norm of the weights multiplied with a constant λ

$$\hat{\beta}^{\mathbf{ridge}} = \mathbf{arg}\min_{\beta} \sum_{\mathbf{i=1}}^{\mathbf{n}} (\mathbf{y_i} - (\beta_{\mathbf{0}} + \beta^{\mathbf{T}} \mathbf{x_i}))^{\mathbf{2}} + \lambda \|\beta\|_{\mathbf{2}}^{\mathbf{2}}$$

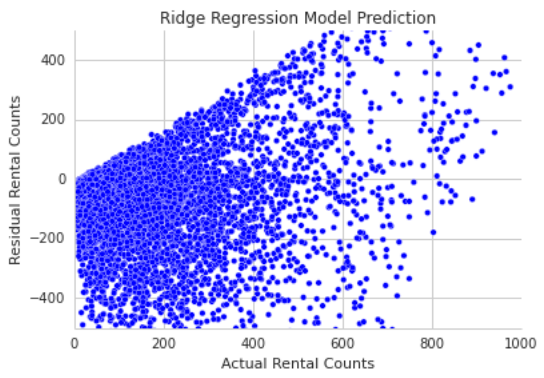


Figure 3.1: Residual plot for Ridge Regression

We got an error(RMLSE) of 1.72

3.1.2 Lasso Regression

Lasso differs from ridge regression in terms of regularization factor. While ridge uses L2 norm, Lasso employs L1 norm on the weights.

$$\hat{\beta}^{\mathbf{lasso}} = \mathbf{arg}\min_{\beta} \sum_{\mathbf{i=1}}^{\mathbf{n}} (\mathbf{y_i} - (\beta_\mathbf{0} + \beta^\mathbf{T}\mathbf{x_i}))^\mathbf{2} + \lambda \|\beta\|_\mathbf{1}$$

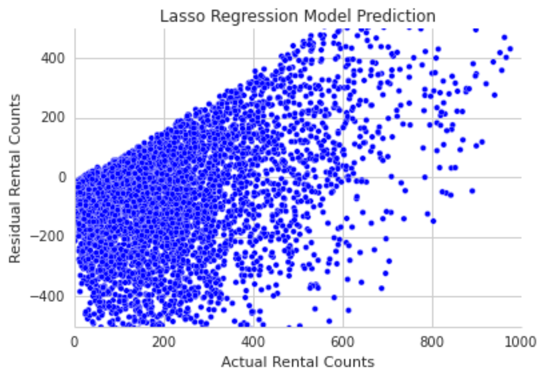


Figure 3.2: Residual plot for Lasso Regression

The error decreased to 1.21 in this case. Next we employed elastic net model on the data.

3.1.3 Elastic Net Regression

Elastic net uses both L1 norm term as well as L2 norm term.

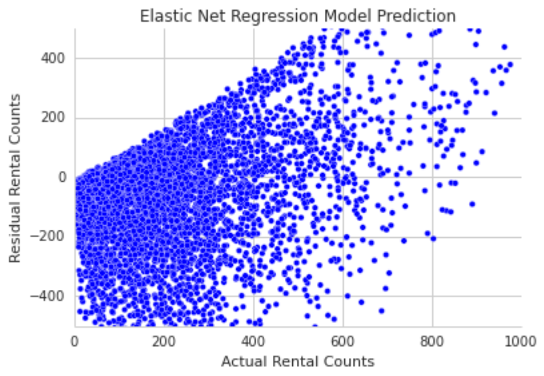


Figure 3.3: Residual plot for Elastic Net Regression

The error in this case was 1.18

3.2 Non-Linear Models

We can see that the performance of linear models on the data was not satisfactory. It suggests that the data is highly non-linear. So we moved on to test some of the non-linear models to capture data in a better way. Here we discuss four non-linear models.

3.2.1 C-Tree

> Conditional inference trees estimate a regression relationship by binary recursive partitioning in a conditional inference framework. Roughly, the algorithm works as follows: 1) Test the global null hypothesis of independence between any of the input variables and the response (which may be multivariate as well). Stop if this hypothesis cannot be rejected. Otherwise select the input variable with strongest association to the resonse. This association is measured by a p-value corresponding to a test for the partial null hypothesis of a single input variable and the response. 2) Implement a binary split in the selected input variable. 3) Recursively repeate steps 1) and 2).[1]

On using this model the RMLSE error suddenly reduced to 0.52. This is a big improvement from linear models. As we can see in the following residual plot, the points are somewhat gathered around $y = 0$ line. So this model provides a better fit to our data.

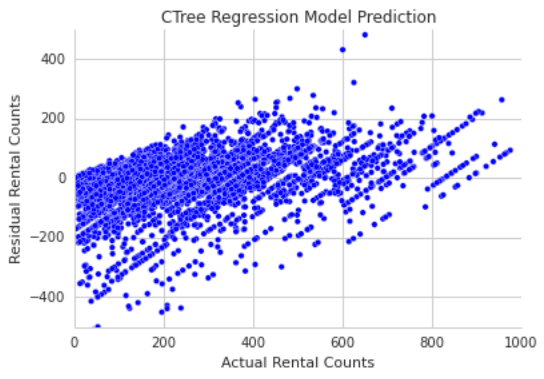


Figure 3.4: Residual plot for C-Tree

3.2.2 Random Forest

We then tried with random forest and this further reduced the error to 0.38. We also tuned the model parameters using grid search in order to select the best possible values for the parameters.

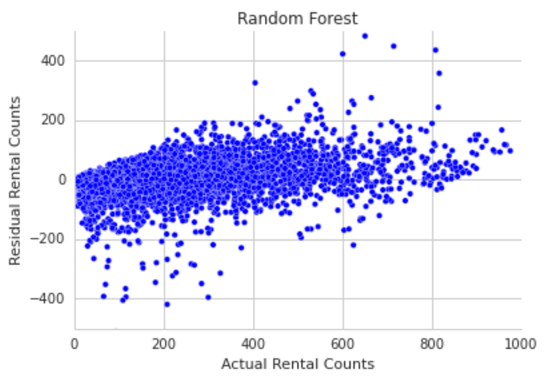


Figure 3.5: Residual plot for Random Forest

3.2.3 Gradient Boosting Model

In gradient boosting machines, or simply, GBMs, the learning procedure consecutively fits new models to provide a more accurate estimate of the response variable. The principle idea behind this algorithm is to construct the new base-learners to be maximally correlated with the negative gradient of the loss function, associated with the whole ensemble.[3]

We tuned the following two parameters using grid search:

- Learning Rate
- Number of estimators(the number of trees)

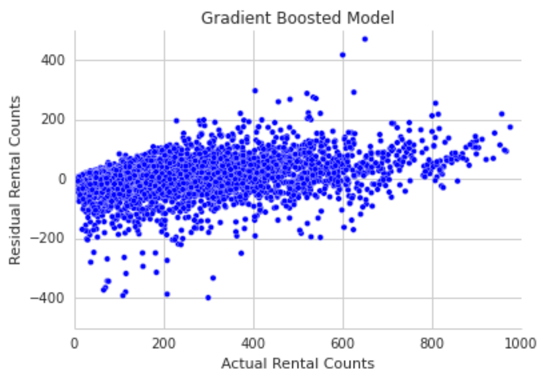


Figure 3.6: Residual plot for Gradient Boosting

We got the lowest possible error of 0.36772 among all the models that we employed. This model took us to 25th rank on Kaggle leaderboard.

3.3 Random Forest with Gradient Boosting

We tried to merge random forest with gradient boosting but it was slightly less accurate as compared to GBM alone.

The RMLSE error in this case was 0.3699

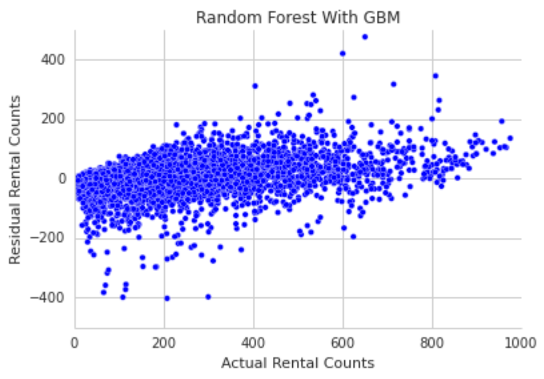


Figure 3.7: Residual plot for RF and GBM combined

4. Other Approaches

We also tried the following approaches

4.1 Ensemble Stacking

As is clear from the name , Stacking is a (common) method of ensemble learning. In stacking several base level classifiers/regressors are trained using the available training data. The predictions of several of these learners using diverse learning algorithms is used as an additional input to the ensembling algorithm known as a meta-learner. The meta-learner can be any learning algorithm , although we have chosen linear regression as our meta-learner , which is very popular. Stacking can be thought of as a non-linear ensembling alternative to voting.

The reasons behind using ensemble stacking is because it often happens that training data doesn't provide sufficent information enough to choose a single learner.In such a scenario combining learners who achieve similar accuracy individually can provide a better approximation[5]. We experimented with GBM , C-Tree , reandom forests as our base learns but none of the ensemblers achieved higher accuracy than our best performing model i.e GBM.

4.2 Neural Net

Neural networks are very popular regression model for learning on non-linear supervised data. First we build our model using the classic back-propagation algorithm. we experimented with the resilent bac-propagation algorithm provided in the 'neuralnet' package in R. We tuned our model by varying the learning rate from (0.3-0.01) and the number of hidden layers (1-10). The predictions however were off the mark in comparison to other tree based methods[4].

In addition to this we experimented with the resilent backpropagation algorithm provided in the 'neuralnet' package in R.It's advantage is that it didn't require any tuning in contrast to the classic backpropagation which is sensitive to learning rate and number of hidden layers and is also faster.

4.3 Extra Tree Regression

Extra-trees closely resemble decision trees . The only difference is the way a split is decided in a more or less randomized way. The $max_features$ attribute (provided by the scikit library) regulates the level of randomization. Setting it to 1 means the split is decided in a completely random matter. For $max - features > 1$, random splits are drawn for each of the randomly selected $max_features$. The prediction of many such randomized extra-trees on various subsamples (number regulated by $n_estimators$) is averaged to improve precision.

4.4 Stochastic Gradient Descent(SGD)

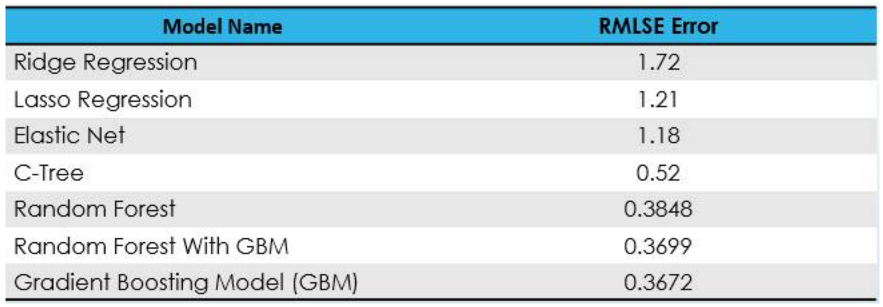
Stochastic Gradient Descent is a simplified gradient descent algorithm in which instead of running through all training to update your parameters in an iteration of the algorithm , a randomly chosed data-point from taining data. It is generally used for data with huge number of training points to improve upon it's complexity.

4.5 Support Vector Regression(SVR)

This is basically an extension of SVM which are generally used for classification. The method which is suitable for data with high dimensionality. The model depends only on a subset of training data because the cost function ignores any training data close to the model prediction. We experimneted with the linear and gaussian kernel functions for the decision function and found the gaussian to be more suited because of the non-linearity in our data.

5. Conclusion & Future Work

We started off with some linear models such as Ridge Regression and Lasso Regression. They did not give satisfactory result. The best accuracy for the linear models that we tried was obtained by Elastic Net Regression(RMLSE=1.18). Then we moved on to non-linear models. Among the tested non-linear models, the best learner for our data came out to be Gradient Boosting Model(GBM) which gave an error of 0.36772.



Model Name	RMLSE Error
Ridge Regression	1.72
Lasso Regression	1.21
Elastic Net	1.18
C-Tree	0.52
Random Forest	0.3848
Random Forest With GBM	0.3699
Gradient Boosting Model (GBM)	0.3672

Figure 5.1: Preformance Evaluation

We also tried some not-so-common approaches such as Ensemble Stacking, Extra Tree Regressor, Support Vector Regressor and Stochastic Gradient Descent. But none of them could surpass the accuracy achieved by GBM. For all these models we also tried to optimize the parameters using gridsearch. The following table summarizes the result of different models that we used.

For future work, one interesting approach that comes to mind is to use C-Trees as base learnes instead of decision trees in random forest. To our knowledge there is no implementation of the same. It would be interesting to see what this model achieves. Next, we only tried linear regressor and support vector regressor as meta-learner in ensemble stacking. So we can experiment with other meta-learners as well.

Lastly, since the count of bikes rented at a particular hour of the day is dependent on the previously seen data we can exploit the idea of time series analysis and see whether this model stands out with others or not.

Bibliography

[1] ctree: Conditional Inference Trees by Torsten Hothorn, Kurt Hornik, and Achim Zeileis. *ctree: Conditional Inference Trees*.

[2] Kaggle Website: Forecast use of a city bikeshare system
https://www.kaggle.com/c/bike-sharing-demand

[3] Gradient Boosting Model: Gradient boosting machines, a tutorial
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/

[4] Neural Network Model: Cran Packages, Neural Net
http://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

[5] Ensemble Stacking: Wikipedia
http://en.wikipedia.org/wiki/Ensemblelearning