Rental Bike Demand Prediction

Litong "Leighton" Dong, Chi-Liang (Daniel) Kuo, and Steven Rea
Advanced Machine Learning
USF

Abstract

Bike sharing systems have gained popularity over the past decade. Because of their accessibility and affordability, more than 500 bike sharing programs are running around the world to provide a low-cost alternative for short-term commutes and to help alleviate traffic. As bike sharing demands increase, unavailability and uneven distribution of bikes become a critical issue for the programs, which could undermine users' experience and waste public resources. In this project, we joined a Kaggle competition to create a better way of predicting the daily bike sharing demands in Washington, D.C. We used random forests (of regression trees) to best predict the number of bikes that will be rented for a given hour. We built our prototype models in R and leveraged Spark to handle massive amounts of data. Our best model so far has reached a pseudo R^2 of 0.96 and our best submission ranked among the top 36% of all models on Kaggle's leaderboard.

Introduction

Bike Sharing System

Bike sharing has been around for 60 years now, and exists in over 500 cities around the world. The systems satisfy a breadth of demands, from leisure activity to commute supplement. They are particularly strong in cities with well-established public transportation, where you only need a bike for that last mile. Given the growth in this area, we wanted to accurately predict the number of bike rentals in a given hour hour while maintaining a generalizable idea that we could apply to other industries.

The problem falls in the domain of regression; of the models available to us, we decided to focus on random forests of regression trees for several reasons, the central one being that we hoped to use this as an opportunity to learn a new technology that could be leveraged in applications in which the amount of data used for analysis or other manipulation does not fit on one machine. Spark is a general engine for large-scale data processing that has gained significant momentum, and we learned how to use it to build random forests to predict bike rental demand.

We looked at data from the largest bike sharing system in the United States to get an idea of what demand will look like for a given hour based on other factors that the company would have available at the time. We started by exploring the data to see if there were any ways to transform or supplement it before moving onto the actual model building phase. We then went through an iterative process of updating our feature set by checking the predictive power of each model before arriving at our final result. After having our final model in mind, we rebuilt and trained it in Spark on Amazon Web Services (AWS).

Related Work

We had several sources of inspiration as we worked through this project:

- This article from our assigned readings had some interesting ideas, one of which we directly applied: http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/
- Professor Interian's demonstration on Spark showed us how accessible this new technology is, and gave us a foundation for how we could use it to accomplish our current goal.
- Many datasets are too large to handle on a single machine, and since AWS is the most popular remote server service, we wanted to make sure we had experience using it.

Dataset Description & Pre-processing

Bikeshare dataset

The dataset is available as part of a Kaggle competition, and was given to us in csv format with 10,886 rows and 12 columns. We performed our analysis in R, and easily imported the data: we simply changed our character columns into factors so that they would be handled correctly throughout the exploratory analysis and modeling.

Daylight dataset

We supplemented the original bikeshare dataset with data from *TimeAndDate.com*. The data available through that site had several columns of interest, such as whether an hour occurred with or without daylight. It was clean and well-structured, which made it simple to merge with the Kaggle dataset.

Spark

The machine learning library we used in Spark, *MLlib*, works better with datasets where categorical variables have already been turned into dummy variables, so we prepared the data in R before exporting it as a .csv file. Once the data was in this format, loading into Spark was very straightforward.

Exploratory Analysis

The bikeshare dataset contains a mix of categorical and numeric variables. The dependent variable is *count*, which is the number of bikes rented per hour. This feature depends on previous values of itself, which led to the first portion of our three-part exploratory data analysis: time series analysis. After we had a better understanding of the autocorrelation, we checked Pearson correlation coefficients between all the numeric features to better understand the interactions taking place there. Lastly, we used box plots to discover any categorical variables with high variance in the dependent variable.

Time Series Analysis

As the dataset is partitioned by hours of the day, we were interested in seeing whether or not any serial correlation existed between values. To investigate this, we plotted the autocovariance function. (See Figure 1.) As might be expected when looking at hourly data, we discovered a pattern that repeated every 24 hours. In addition, we performed a Box-Pierce test which confirmed that we have statistically significant lags. Therefore, we decided to incorporate several lag terms into our features, mimicking the process of a Box-Jenkins model.

Correlation

Among our numerical variables, we noted which were highly correlated among each other (such as temperature *temp* and "feels like" temperature *atemp*) to test later during the model-building phase. We assumed that if we had two predictors that were very similar in nature but one outperformed the other that there would be no need to hold onto the weaker predictor for our final set of features. Furthermore, the sum of *casual* and *registered* is equal to the dependent variable *count*, which could be a very strong indicator in our models. Nonetheless, the testing set does not include *casual* or *registered*. Hence, we dropped them from our list of features to test.

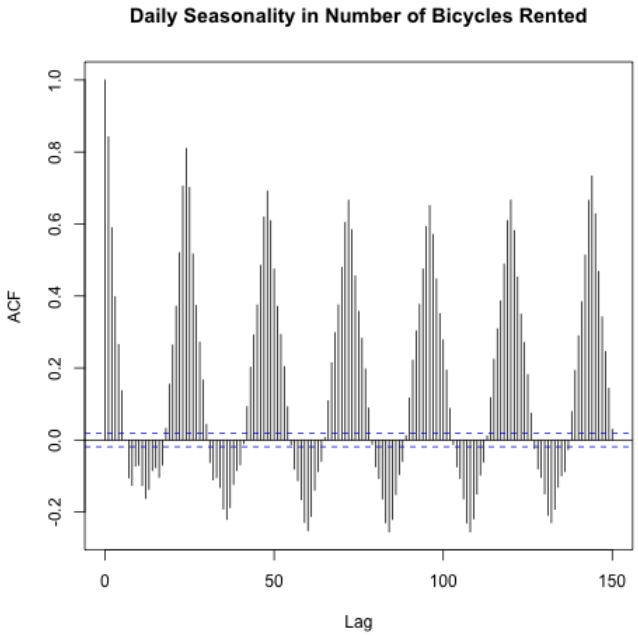

Figure 1: A plot of the Autocorrelation Function, showing serial correlation in the dependent variable *count*.

Preliminary Feature Selection

To investigate the relationship between quantitative output and the categorical variables, we produced multiple box plots for each categorical variable to measure central tendency and spread of the dependent variable, and also to detect potential outliers. The most interesting discovery was found in the boxplot of *hour*. (See Figure 2.) The strong deviation in central tendency at each hour suggests it could be a powerful predictor.

Features

As we examined the data given to us, one of the first things that came to mind was to split the datetime into several different fields. We wanted to be able to look at the data from different points of view and give the algorithms we were going to use the same advantage. As we split the datetime field we noticed one of the pieces of information would be useless for our purpose: Kaggle split the training data and test data by day of the month, with the first 19 days of each month for two consecutive years making up the training data and all the rest making up the test data. So, the test set would not have any of the same dates as the training set (20-end vs. 1-19). We noted this and decided that it would be the first feature we drop from consideration during the feature selection portion of the analysis.

After looking through the data, there were several different partitions that were currently not present but that

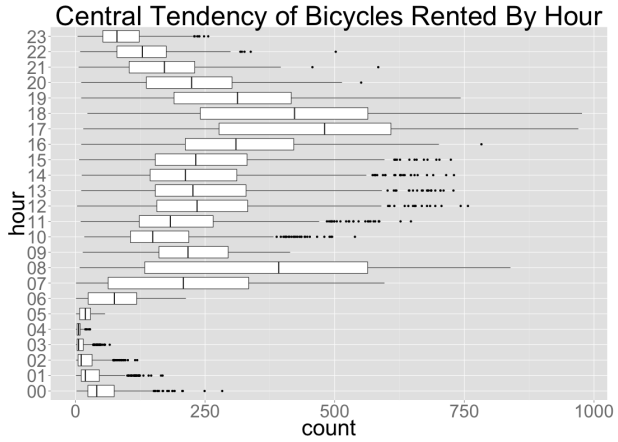

Figure 2: Variance in *count* (of bicycles rented) by hour of the day.

we thought would be interesting to see, and which we hoped might lead to more predictive power. One set of data we thought would be useful to add was related to sunlight, so we downloaded data related to the amount of daylight in Washington, D.C. over the testing period. The dataset included times of sunrise and sunset, and the number of hours of daylight there were in the day.

After importing the additional data and using it along with the original dataset, these were the most important extra features added:

- *is.daylight*: Boolean for whether daylight is present (based on sunrise and sunset).
- *is.working.hour*: Boolean for whether the hour takes place between 9 and 5, when most people are at work.
- *day.part*: Feature with character data representing a partition of the day into four character values: "Morning", "Midday", "Afternoon", and "Night".
- Lagged Counts: Counts from previous hours (1,2,3, and 24 hour lags).
- *diff*: The difference between the previous two hours' counts.

In adding lagged hours, there were instances where we did not have earlier data (e.g. for the first day of data, there is no data for 24 lags in the past). The random forest algorithm in R does not allow missing values so by adding these features we were omitting observations. This is something we noted during the feature engineering process as something to check while performing feature selection. The question we kept in mind was: Is the information gained from the extra feature worth the information lost from the omitted observations?

Machine Learning

Feature Selection

The variable to be predicted is the number of bikes that would be rented in an hour. We chose to solve this problem using regression trees and random forests. Random forests are created by "growing" a forest of trees. Each tree in the forest is created from a random sample of the original dataset with replacement. So, each tree has a different group of the data that it understands. As the tree is grown, splits in the tree come from a random subset of the full feature set. When working with an individual tree, you use the entire dataset rather than a sample of it, and you consider all features throughout the process. Otherwise the two models are very similar to one another.

Random forests have characteristics that make them desirable from a modeling perspective. In particular, because a forest is made up of many trees, the overall variance is reduced compared to individual trees. On the other side of the bias-variance tradeoff, the algorithm keeps the small bias of a tree while reducing variance. Furthermore, with many trees, the algorithm is more robust to the noise in a given dataset when compared to other algorithms, which helps increase its accuracy. From a model tuning perspective there are few parameters to tune, and as we found out later in the analysis, the default parameters served us very well. Although it does not apply to our problem, the algorithm is suitable even when the number of predictors is much greater than the number of observations.

On the other hand there are some downsides to the algorithm which we kept in mind during the model selection phase and throughout the experiments. In particular, random forest is a less interpretable model. You are not given a nice interpretation of how much each feature directly affects the dependent variable like you are in Linear Regression. Since you are growing many trees, it can take much longer than other algorithms. Some things that we kept in mind regarding the trees themselves are that you cannot predict beyond the range seen in your training data, and, similarly, extreme values are often predicted with much less accuracy.

We were more concerned with predictive power than interpretation; the variable importance plot given in R's randomForest package was sufficient to indicate which features were most influential in creating pure subsets of the data. And even though the models are slower to train, we were not concerned with speed because we were going to distribute the growing of the trees over many nodes in a cluster. The bagging technique used by random forests lends itself well to distributed computing (unlike other techniques such as boosting) because the bootstrapped trees are grown independently of one another. Extreme values were the only concern we had from the list of weaknesses in the algorithm.

To compare the different random forests we looked at out-of-bag error, defined as "the mean prediction error on

each training sample x_i, using only the trees that did not have x_i in their bootstrap sample" in *An Introduction to Statistical Learning* by James et al., as well as a pseudo R^2 metric, calculated as $1 - MSE/Var(y)$.

We made sure to only remove one feature at a time so that we did not overlook any interactions that may have been taking place. If a variable we dropped negatively affected the model, we added it back in. Otherwise we left it out.

Our first idea was to drop *day*. From a purely mathematical point of view, none of our training data will have the same day of the month as our testing data, so training a model based on something we know will not be helpful if used does not make sense. From a more intuitive approach, the day of the month has no bearing on whether someone will do something unless it is a holiday, which would be a yearly occurrence captured by the *holiday* feature.

While checking the variable importance plots after the previous iteration we noticed that holiday contributed little to the predictive power of the algorithm, so we tried dropping it. But, even though its predictive power was not as high, it was a noticeable loss of accuracy, so we left it in the model.

We then moved on to see whether the observations lost from incorporating the lagged variables were worth having the extra features. The model's errors were significantly higher without them so we decided to keep them in. While checking the correlation among our predictor features we noticed that *atemp* and *temp* are strongly correlated, with *temp* being more highly correlated with *count*. Dropping *atemp* led to slightly better results, so we kept it out. Checking correlations again led to us trying to drop *count.2* and *count.3* since they were so highly correlated with *count.1* but *count.1* is most correlated with the dependent variable, *count*.

Next we wanted to test removing *temp* and *day.hours* since they are highly correlated. *Temp* has stronger predictive ability, so we dropped *day.hours*. As we saw before, two of the lagged features, *Count.1* and *count.24*, are highly correlated. *Count.1* has stronger correlation with *count*, so we removed *count.24*, which led to significantly better scores.

While working with the data we had the idea to add two extra features: *diff* and *accel*. *Diff*, short for "difference", is the change in number of rentals from two hours ago to the previous hour. *Accel*, short for "acceleration", is the change in the change of the previous two sets of hours.

Having both of the new features ended up hurting the predictive ability, but once we removed *accel* we were able to bring our predictive ability above where it was without *diff*. The next weakest predictor was *windspeed*. But removing it hurt the model, so at this point we decided to take another look at everything we had.

While looking at this model, we thought of factoring the dates by day of the week would improve the model. In particular we were curious whether or not a boolean flag to determine whether the day of the week is Sunday would help us achieve more significant results. We decided to add each variable independently to see how they affected our then-strongest model.

Is.sunday was not nearly as useful as *weekday*, and it did not make sense to have the redundant information in the model, so we removed it from our set of strongest features. *Weekday* on the other hand ended up significantly contributing to an increase in accuracy, and was the final addition to our strongest model. So, our final model was a random forest with the following features: *count.1*, *day.part*, *diff*, *holiday*, *hour*, *humidity*, *is.daylight*, *is.working.hour*, *month*, *season*, *temp*, *weather*, *weekday*, *windspeed*, *workingday*, and *year*. (See Figure 3 for a comparison of our the features that best split the data into the purest subsets.)

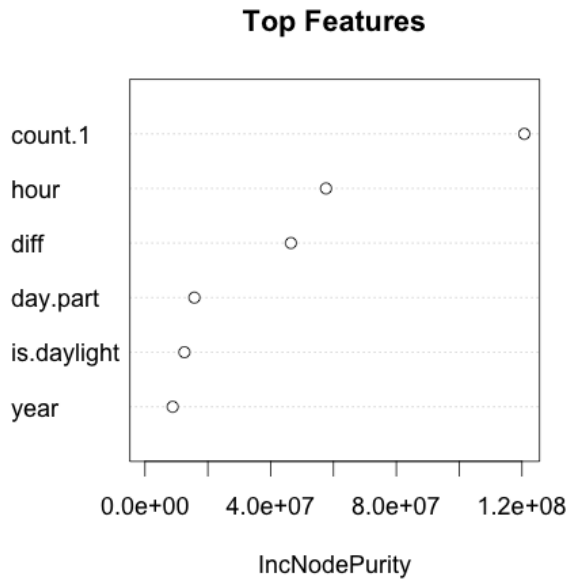

Figure 3: Variable Importance plot of the top features from our final model. The one hour lag became the strongest feature.

Hyperparameter Tuning

After feature selection, we performed hyperparameter tuning on our random forest model. Specifically, we varied the *number of trees* and *number of features considered per split* to optimize over the validation metrics. As you can see in Table 4 in the Appendix, as we increased the number of features from 1 to 11, the MSE first decreased dramatically and reached a minimum around 5 or 7, and then started to increase again when the number of features grew to be more

than 9. This offers more proof of the common heuristic to choose the square root of the number of features for this hyperparameter. Next we looked to adjust the number of trees in the forest: as the number of trees grew to more than 300, the out-of-bag MSE reached a minimum along this dimension. Therefore, after we tried several variations without observing any significant improvements of other combination of hyperparamters, we decide to retain the original one: 5 random features considered per split and 500 trees in total.

Results

Our final model, on average, missed 25 bikes per hour. To give some perspective, there were, on average, 191 bikes rented per hour throughout the city. And, a random forest model with default hyperparameters and just the original dataset with no feature engineering missed 55 bikes per hour.

One of the more surprising results from this was comparing the error reduction that happened in two particular phases of the project. If you think of three points in time:

1. Raw Data: This represents the data as given to us by Kaggle, before we added any new features.
2. After Feature Engineering: This represents the extra features we added while looking at the data, and before removing any weak predictors.
3. After Feature Selection: This represents our final feature-set, after making sure we removed all of the weakest predictors.

There was far more error reduction in the model after the feature engineering phase than after the feature selection phase. (See Figure 4.) This initially surprised us, but made sense given previous projects we have worked on with the random forest algorithm. In one prior project we had far more features but feature selection did not end up helping the final results as much as it did in this analysis.

In order to make our prediction without violating Kaggle's rules, we trained different versions of our final model for each month within the historical data accumulated. This came down to training one model in the first month, then one model on the first two months, etc., until we had twenty four different models, one to predict for each of the months of test data. Once the models were trained, we predicted the hourly demands for the rest of the days of each month. Based on this framework, we made five submissions to the competition. Our best submission was the final one, which ranked among the top 36% of all submissions on the Kaggle leaderboard. (See Table 1 for error score by submission.)

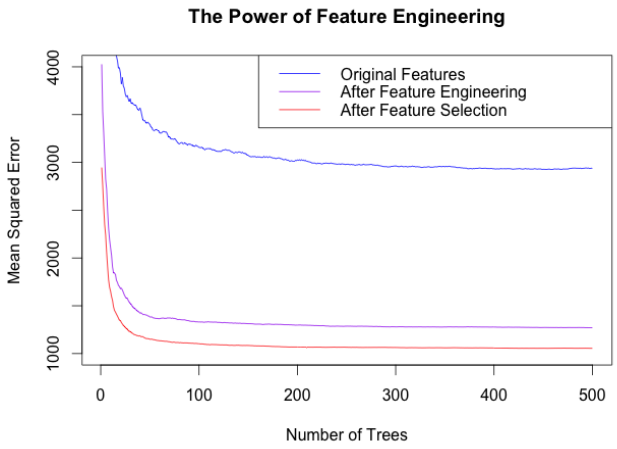

Figure 4: The reduction in MSE after feature engineering and then after feature selection.

Kaggle Submission Error Rates

	1	2	3	4	5
score	0.847	1.280	0.494	0.496	0.484
model	R.T	R.T.	R.F	R.F.	R.F.

Table 1: This table shows the different errors we obtained through our various submissions to Kaggle. (R.T. for regression tree and R.F. for random forest)

Your challenges and successes

Spark

One of the motivations of this project was to learn how to set up and use a Spark cluster. With this in mind, at the beginning of this project, we decided to explore the use of Apache Spark not only on a local machine, but also on a virtual cluster. Spark is a cluster computing system similar to Hadoop. The idea behind cluster computing is that, if there are a large number of tasks to be executed, and they are independent of each other, they can be executed on many different computers and their results can be combined at the end. This idea is called parallel processing and can be completed by using Map-Reduce operations. The key difference is that Spark keeps the data in memory instead of on disk, allowing it to achieve better performance than Hadoop.

Although Spark is more power than needed for our small dataset, we believed that the experience and skills we would learn from this would be transferred into our future work, where we will likely encounter very large datasets. We planned to include a beginner-friendly, step-by-step tutorial on how to launch and use Spark on Amazon Web Service (AWS) EC2 virtual Linux machines, in addition to our presentation and report. Preparing the tutorial for the reader helped us by solidifying the concepts in our minds; the best way to learn is to teach.

The very first challenge we encountered was obtaining and setting up the security credentials for launching EC2 machines. Such preparation was required to remotely access (ssh) the virtual machines. One key difference is that before we accessed the machines, we used a script provided by

Spark to create these virtual machines via command line instead of through Amazon's website. This obstacle was relatively easy to overcome.

After we launched and logged into our Spark cluster, we immediately encountered our second challenge – the unique data structure of Spark. Spark uses Resilient Distributed Datasets (RDDs), which are created from transformed files in Hadoop-style distributed file systems. RDDs can be stored in memory for fast operations and are immune to node failures. They use Map-Reduce operations like other Hadoop-style distributed files. Lacking tim, experience with Map-Reduce operations, and the familiarity of Spark commands, we struggled to perform our exploratory data analysis on Spark. In the end, we decided to continue our exploratory data analysis locally in R and started exploring the Machine Learning Library (MLlib) on Spark.

The documentation for the machine learning library in Spark is relatively unorganized, but still manageable. We were able to find help on the functions and arguments we needed, and the main challenge was to get the data ready in the format required by the library. The functions in the library require input data in "labeled point" format, which is a tuple with the response/target at the first position, and the feature vector at the second position. However, the feature vector only accepts numeric values. Our dataset contains categorical features for which each label needs to be transformed into a dummy variable. We started by trying to do this in Spark. However, due to the difficulty of data transformation in RDDs, we created dummy variables for our categorical features in R instead.

In the end, we were able to successfully launch Spark clusters on 20 of Amazon Web Service's EC2 virtual machines, log in to the master machine, upload our data and run our script. In our script, we first split the data into 40 partitions and stored them in memory. Then, we used functions in MLlib to build a regression tree model and a random forest model. We performed some hyperparameter tuning on the number of trees and the maximum depth of each tree without using an exhaustive grid search. Mean square error (MSE) was our main metric for performance measurement, but we also computed a pseudo R^2 for display purpose. We tried different combinations of total number of trees (500, 1,000, 1,500, and 2,000) and maximum depth of each tree (6 to 12). Both training and testing mean square error were computed. We discovered that as the maximum depth of trees increased, the training error decreased, but the testing error increased as well, which led us to conclude that we were overfitting. We also found out that the total number of trees had no significant effect on error reduction once the size of the forest was greater than 500. Therefore, we concluded that 500 trees and a maximum depth of 8 layers led to the strongest random forests. Similarly, a maximum depth of 8 layers led to the strongest predictions for a single regression tree. The final MSE and pseudo R^2 are shown in Tables 2 and 3, respectively.

	Random Forest	Regression Tree
Training	1,079	1,326
Testing	1,952	2,914

Table 2: Mean Square Error (MSE) Comparison

	Random Forest	Regression Tree
Training	0.9683	0.9610
Testing	0.9399	0.9104

Table 3: Pseudo R^2 Comparison

Kaggle Rules

The rules for the Kaggle competition were not as intuitive as they could have been. Predicting eleven days into the future without updating your inputs as you get more information is unrealistic. We were able to feed our predictions back into the model as inputs, but it added a step of uncertainty that will probably never be present in a real-world application.

Also, we were hoping to compare our model against what others had done, but because these rules were not fully understood by contestants, the leaderboard contains submissions from people predicting the past from the future. Even so, we were still able to score higher than the Mean Value Benchmark in this Kaggle competition.

Extensions and Business Applications

In a real-life scenario, assuming that a business can act on information within one hour might be unrealistic, and varies greatly from domain to domain. However, exchanging the one hour lag for the twenty-four hour lag would not significantly reduce the model's accuracy, and would give a good amount of time to act on the information presented.

Had this been a problem we were working with the company to solve, we would have asked how quickly they could get information on the number of people registered for a given hour. Throwing out that feature did not seem like the correct route to take.

The original dataset contained more features, including the location of bike stations from where the bikes were rented and to where they were returned. This kind of geospatial data can add much more complexity to the model, but could make the results more actionable to the existing bike sharing programs. Including such information would be immensely important were we looking to hand this off for implementation. Otherwise we would only be able to comment on the total number of bicycles needed to be rented cumulatively in any given day.

Conclusions

Random Forests vs Regression Tree

Being able to combine the advantages of both complex and simple model is a key to succeed this project. At the beginning stage, we used a regression tree to expedite the training and prediction process, giving us quick feedback of how each feature performed and what caused the model to perform poorly. Thanks to the regression tree's interpretability and simplicity, we were able to adjust our set of features by adding more powerful features and removing the less effective ones. At this point it was easy to see how they affected the evaluation metrics. (See Submissions 1 and 2.) After we added more features and better understood the data, we used random forests to boost the accuracy we were able to get using regression trees. (See Submissions 3 and 4.) Overall random forests once again showed strong predictive power on a dataset that was different from others that we have worked with.

Feature Engineering

We have heard about the relative importance of feature engineering, but seeing it affect our results to much in practice has strongly reinforced this idea in our minds. The comparative advantage shown in Figure 4 really underscores the importance of taking time with your data before you consider what algorithms will work best for your application. Looking at our fourth and fifth submissions as examples, the main difference between these two models is that we included *weekday*, a factor of the day of the week, into our model because we found the average number of bikes rented to be quite different on each day of the week. In addition to taking new looks at your existing data, the extra insights we were able to get from merging data from outside the competition also proved to be a good lesson for us. The great results we were able to achieve has taught us to always look around for supplemental data when solving a problem.

Furthermore, we learned that the removal of correlated columns can noticeably increase the predictive ability of the model. Previous projects we worked on have not had such a noticeable improvement from feature selection. (Though this may be due to the fewer features available for this dataset.) The idea originally came from our work with Linear Regression and our knowledge of Linear Algebra, so it was very interesting to see the results of this in this different context.

Spark

The potential of Spark's distributed computing was not fully realized in our case, with the small dataset we were working with. Nonetheless, we went through the same process of any project in Spark, obtained two models, and computed error respectively. The hardest part of any new technology is getting it set up without the assistance of organized documentation. To overcome this obstacle, it required time and effort in familiarizing ourselves with the technology. With the experience gained through our exploration, we will now be able to apply this power more easily in our future projects.

Appendix

Spark Tutorial

Throughout our report we kept track of what we were doing in order to share a tutorial with others who might be interested in learning how to set up Spark on an EC2 node. The tutorial, as well as our code, can be found in our Github bucket:
`https://github.com/ldong7/Bike-Share_Demand_Public`

R EDA & Prototyping

This github repository included all the data pre-processing, transformation, EDA feature selection, hyperparamter tuning, and OOB error measurement.
`https://github.com/ckuo7/bikeSharingDemand`

Initial Variables

The initial variables from the dataset were

- Count (dependent variable): Total number of bicycles rented in a given hour.
- Registered: Number of non-registered user rentals initiated.
- Casual: Number of non-registered user rentals initiated.
- Datetime: Hourly Date with timestamp.
- Windspeed: Wind speed.
- season: Season of the year, represented as 1 (Spring) through 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.

(continue to next page)

Hyperparamenter Tuning

Number of Trees\Number of features	1	3	5	7	9	11
100	4905.691	838.7317	784.7302	781.0359	783.2215	806.374
300	4320.188	834.8003	788.3556	774.1092	783.9893	804.3188
500	4850.806	836.3793	789.6741	782.903	782.9469	800.7366
700	4466.266	828.5678	781.9912	782.2162	793.5692	799.0603
900	4921.609	842.1087	781.1956	775.463	788.328	804.2761

Table 4: Hyperparamter Tuning