Bike Sharing Demand Kaggle Competition: A Feature Engineering Approach

Jose Millan
jampmil@gmail.com
Data Mining & Knowledge Management Masters Program
Università degli Studi del Piemonte Orientale 'Amedeo Avogadro'
Alessandria, Italy

Abstract

Between May 2014 and May 2015 the Bike Sharing Demand Kaggle Competition took place for the Capital Bikeshare program in the city of Washington, D.C. Focusing in feature engineering, the proposed work presents a python-based solution for this competition. Proposing new features and using an ensemble model, this solution is able to land within the Top 100 entries in the Kaggle's Leaderboard, with a RMSLE of 0.37466.

1 Introduction

Between May 2014 and May 2015 the Bike Sharing Demand Kaggle Competition [2] took place for the Capital Bikeshare program in the city of Washington, D.C. Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. The objective of the competition in question is to predict the total count of bikes rented on an hour basis, based on the training data provided.

The current work presents a python-based solution for this competition, using different regression models and, above all, focusing in feature engineering based on the given dataset. The presented solution is able to predict the count of rented bicycles with a low RMSLE, and is ranked between the top 100 best solutions according to Kaggle's leaderboard.

2 Given Datasets

For the competition, two datasets are provided with the information about hourly rental data spanning two years, where the training set is comprised of the first 19 days of each month, while the test set is the 20th to the end of the month. The training dataset contains the following features:

- `datetime`: Timestamp containing the hour and date in the form: `YYYY-MM-DD hh:mm:ss`
- `season`: Current season of the record. The possible values for this feature are:
  1: Spring
  2: Summer
  3: Fall
  4: Winter
- `holiday`: Whether the day is considered a holiday.
- `workingday`: Whether the day is a working day (neither a weekend nor holiday).
- `weather`: Current weather of the record. The possible values for this feature are:
  1: Clear, few clouds or partly cloudy.

2: Misty and cloudy, misty with broken clouds, misty with few clouds or misty.

3: Light snow, light rain with thunderstorm and scattered clouds or light rain with scattered clouds.

4: Heavy rain with ice pallets, thunderstorm and mist or snow with fog.

- `temp`: Current temperature in Celsius.
- `atemp`: Current "feels like" temperature in Celsius.
- `humidity`: Current relative humidity.
- `windspeed`: Current wind speed.
- `casual`: Number of non-registered user rentals started.
- `registered`: Number of registered user rentals started.
- `count`: Total number of rentals. Equals to the sum of casual and registered.

3 Kaggle's Submission and Evaluation

For this competition, and in order to participate in it and get a scored value of the submission, a CSV file has to be generated containing the respective timestamps present in the test set along with the correspondent total number of rentals prediction. Using this file, Kaggle evaluates the submissions using the Root Mean Squared Logarithmic Error (RMSLE), as defined in Formula 1,

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

where n is the number of hours in the test set, p_i is your predicted count, a_i is the actual count and $log(x)$ is the natural logarithm.

For consistency, the same measure of error in the predictions is going to be used for the purpose of this work.

4 Dataset Exploration

The first step to explore the given dataset is to obtain some simple descriptive statistics from it, which can be seen in Figure 1, along with their graphical representation in Figure 2.

Features Types

casual	float64
registered	float64
atemp	float64
humidity	float64
temp	float64
windspeed	float64
weather	float64
season	float64
workingday	float64
holiday	float64

	casual	registered	atemp	temp	humidity	windspeed	weather	season	workingday	holiday
count	10886.0	10886.0	10886.0	10886.0	10886.0	10886.0	10886.0	10886.0	10886.0	10886.0
mean	36.021955	155.552177	23.655084	20.23086	61.886460	12.799395	1.418427	2.506614	0.680875	0.028569
std	49.960477	151.039033	8.474601	7.79159	19.245033	8.164537	0.633839	1.116174	0.466159	0.166599
min	0.000000	0.000000	0.760000	0.82000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
25%	4.000000	36.000000	16.665000	13.94000	47.000000	7.001500	1.000000	2.000000	0.000000	0.000000
50%	17.000000	118.000000	24.240000	20.50000	62.000000	12.998000	1.000000	3.000000	1.000000	0.000000
75%	49.000000	222.000000	31.060000	26.24000	77.000000	16.997900	2.000000	4.000000	1.000000	0.000000
max	367.000000	886.000000	45.455000	41.00000	100.000000	56.996900	4.000000	4.000000	1.000000	1.000000

Figure 1: Basic Statistics of Given Features

From these statistics it is important to notice some things:

- In average, there the number of registered rentals is more than 4 times the number of casual rentals. Additionally, there is a considerable number of outliers for both casual and registered rentals.

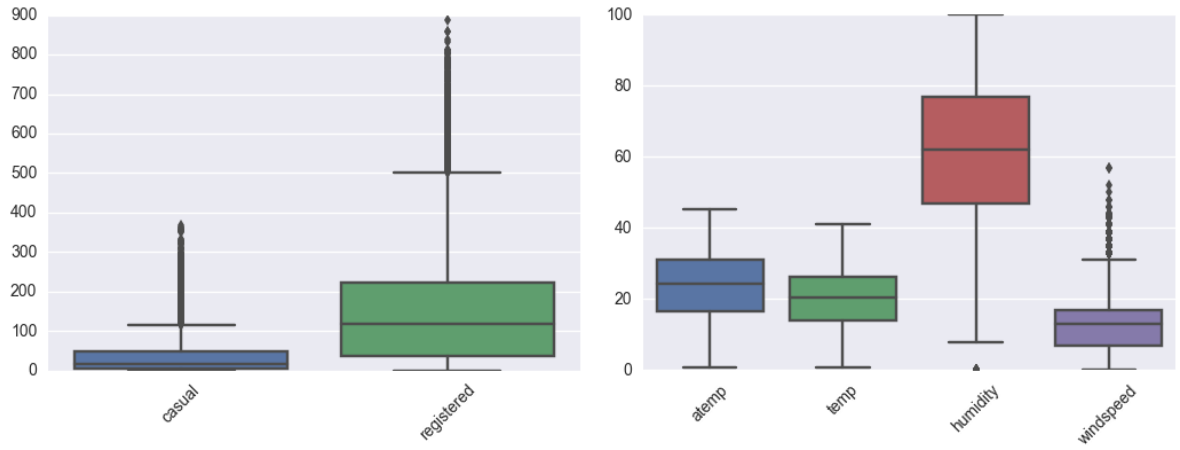

Figure 2: Boxplot of Basic Statistics of Given Features

- The "feel like" temperature is in average three degrees greater than the temperature given; nevertheless both measures are distributed fairly similar.
- Wind speed presents a high number of outliers. This can make sense if strong winds are present sporadically when there are storms or hurricanes on the east coast.
- The weather is clear for more than half of the records (`weather = 1`). Considering this, it is possible that predicting the number of rentals on bad weathers present a greater challenge.
- The seasons appear to be equally distributed on its four values. This is an indicator that we have a complete sample for the 2 years.

Additionally, in Figure 3 a correlation matrix of the given features can be appreciated. Based on this matrix it can be seen that the temperature is highly correlated with the number of rentals, whereas the humidity is negatively correlated with them. Additionally, it is also interesting to notice that the working day indicator is negatively correlated only to the casual rentals and not to the registered, which means that casual rentals are more common on non-working days (weekends and holidays) rather than in working days, whereas this does not have a big impact on the registered rentals.

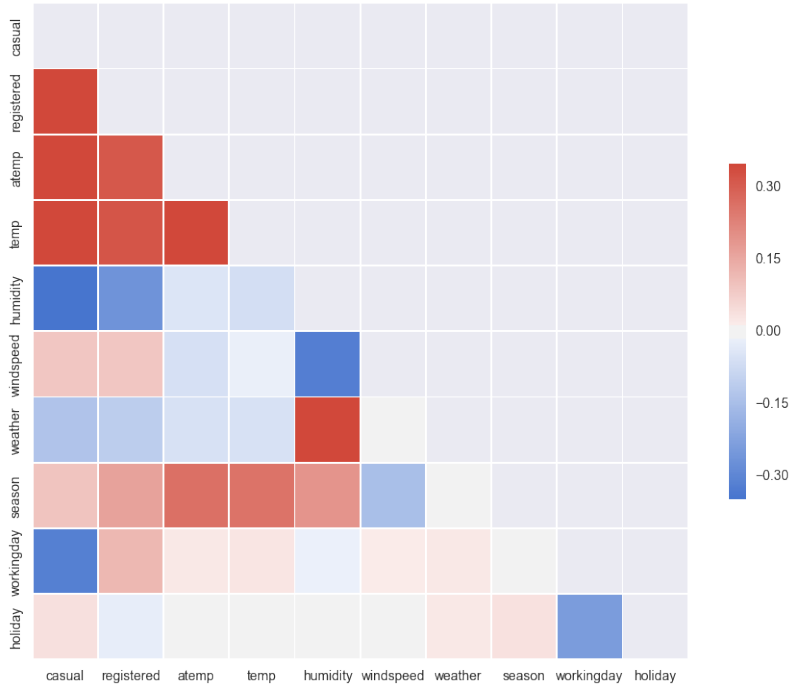

Figure 3: Correlation Matrix for Given Features

5 Feature Engineering

Based on the features given, the following feature were thought to be useful in the current work. It is important to mention that all the features considered are being presented below, nevertheless not all of them are used in the final model, being carefully selected as described in Section 6.

5.1 Time Features

From the `timedate` feature included in the given dataset, the following features were derived:

- `hour`: Hour of the day. An integer between 0 and 24.
- `day`: The day of the week. An integer between 0 and 6 where 0 is Monday and 6 is Sunday.
- `week`: The week of the year. An integer between 1 and 53.
- `dayOfMonth`: The specific day in the timedate. An integer between 1 and 31.
- `month`: The specific month of in the timedate. An integer between 1 and 12.
- `year_quarter`: A number specifying the quarter of the year. An integer between 1 and 4.
- `year`: The specific year. An integer with values in [2011, 2012].
- `weekend`: Specifies if the current date is part of the weekend. `weekend` is `1` if `day = 5 or day = 6` and `0` otherwise.

5.2 Weather/Environment Features

Three features were constructed based on the weather conditions or environmental events that happened during the years of the given data.

- `rain`: A variable that indicates if it is currently raining. `rain` is `1` if `weather = 3 or weather = 4` and `0` otherwise.
- `sticky`: A variable that indicates if the current weather conditions can be considered as "sticky". This is simply defined as a weather with high temperature and high humidity. `sticky` is `1` if `humidity >= 70 or temp = 26` and `0` otherwise.
- `nat_dis`: A new variable constructed from external information [5], which indicates if at the current moment a natural disaster was happening. The specific natural disasters that were taking into consideration were:

  North American blizzard: Jan 8 – 13, 2011.

  Earthquake: Aug 23, 2011.

  Hurricane Irene: Aug 21 - 30, 2011.

  Storm: May 21, 2012.

  Tornado: Jun 1, 2012.

  Hurricane Sandy: Oct 27 - Nov 1, 2012.

5.3 Rush Rentals Features

Considering the idea users rent more bicycles in specific hours rather than others, a new feature that represented this rush hours was included. This idea was then expanded not to be just about the rush hours but to use the same concept for rush temperatures, rush humidities and rush wind speeds. It is important to understand that since the nature of the causal and registered rentals are different, different rush features were implemented for both of them.

For each one of these features, two different methods were used to assign the indicator. This means that for each feature (hour, temperature, humidity and wind speed) a set of four new features was added.

- A manual process where, by studying the distributions between the feature and the number of rentals (for casual and registered separately), the best segments of the features were found.
- An automatic process where a simple decision tree was built between the feature and the number of rentals to generate the decision rules that separates the feature in different buckets.

5.3.1 Rush Hours

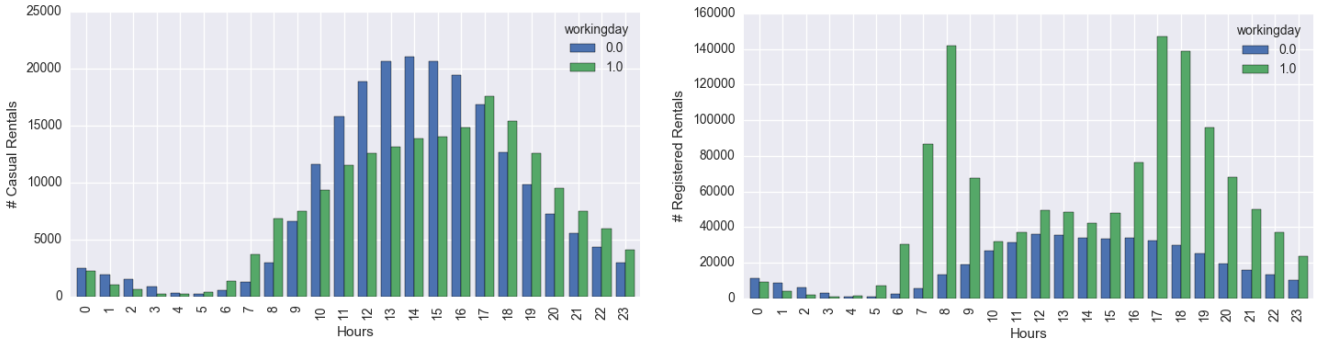

Figure 4: Histogram Hour vs Number of Rentals

Based on the histograms of the casual and registered hours (as seen in Figure 4), the following rules were used to assign the values of the new `rush_hour_casual_man` and `rush_hour_reg_man` features:

- Casual Rentals:
  `rush_hour_casual_man is 1 if`
  `(working_day = 1 and 7 <= hour <= 23)`
  `or (working_day = 0 and 10 <= hour <= 19)`
  `0 otherwise.`
- Registered Rentals:
  `rush_hour_reg_man is 1 if`
  `(working_day = 1`
  `and (6<= hour <=9 or 16<= hour <=23))`
  `or (working_day = 0 and 8 <= hour <= 23)`
  `0 otherwise.`

For the automatic rush hour buckets, the respective rules that were generated by using a decision tree, with $depth = 2$, can be appreciated in Figure 5.

```
def rush_hour_casual_bucket(hour):
  if hour <= 9.5:
    if hour <= 7.5:
      return 1
    else:  # if hour > 7.5
      return 2
  else:  # if hour > 9.5
    if hour <= 19.5:
      return 3
    else:  # if hour > 19.5
      return 4
```

```
def rush_hour_reg_bucket(hour):
  if hour <= 6.5:
    if hour <= 5.5:
      return 1
    else:  # if hour > 5.5
      return 2
  else:  # if hour > 6.5
    if hour <= 20.5:
      return 3
    else:  # if hour > 20.5
      return 4
```

Figure 5: Decision Rules for Rush Hour Automatic Buckets

5.3.2 Rush Temperature

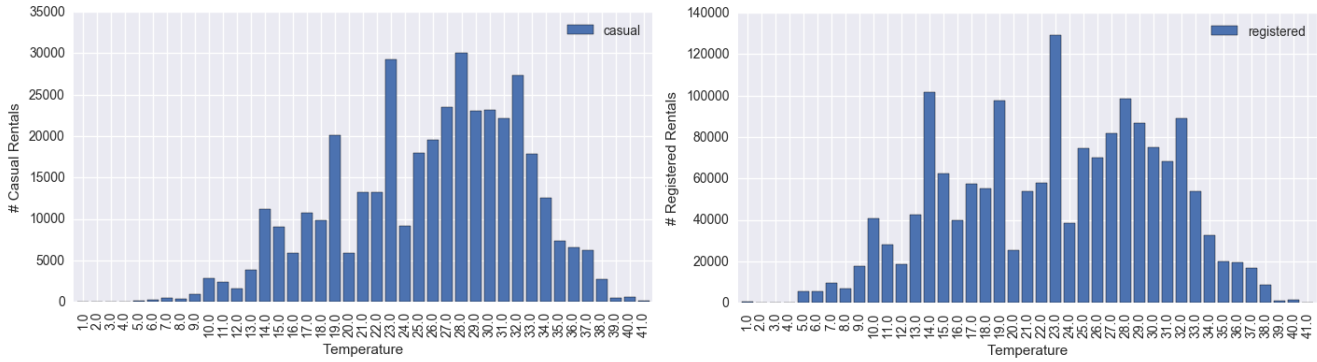

Figure 6: Histogram Temperature vs Number of Rentals

Based on the histograms of the casual and registered rentals per temperature (as seen in Figure 6), the following rules were used to assign the values of the new `rush_temp_casual_man` and `rush_temp_reg_man` features:

- Casual Rentals:
  `rush_temp_casual_man is 1 if`
  `(18.5 <= temp <= 19.5)`
  `or (22.5 <= temp <= 33)`
  `0 otherwise.`

- Registered Rentals:
  `rush_temp_reg_man is 1 if`
  `(13.5 <= temp <= 15)`
  `or (18.5 <= temp <= 19.5)`
  `or (22 <= temp <= 23.5)`
  `or (27 <= temp <= 32)`
  `0 otherwise.`

For the automatic rush temperature buckets, the respective rules that were generated by using a decision tree, with $depth = 3$, can be appreciated in Figure 7.

```
def rush_temp_casual_bucket(temp):
  if temp <= 23.3699989319:
    if temp <= 15.1700000763:
      if temp <= 12.7100000381:
        return 1
      else:  # if temp > 12.7100000381
        return 2
    else:  # if temp > 15.1700000763
      if temp <= 19.2700004578:
        return 3
      else:  # if temp > 19.2700004578
        return 4
  else:  # if temp > 23.3699989319
    if temp <= 29.9300003052:
      if temp <= 29.1100006104:
        return 5
      else:  # if temp > 29.1100006104
        return 6
    else:  # if temp > 29.9300003052
      if temp <= 32.3899993896:
        return 7
      else:  # if temp > 32.3899993896
        return 8
```

```
def rush_temp_registered_bucket(temp):
  if temp <= 22.5499992371:
    if temp <= 12.7100000381:
      if temp <= 11.0699996948:
        return 1
      else:  # if temp > 11.0699996948
        return 2
    else:  # if temp > 12.7100000381
      if temp <= 19.2700004578:
        return 3
      else:  # if temp > 19.2700004578
        return 4
  else:  # if temp > 22.5499992371
    if temp <= 29.9300003052:
      if temp <= 28.2900009155:
        return 5
      else:  # if temp > 28.2900009155
        return 6
    else:  # if temp > 29.9300003052
      if temp <= 30.75:
        return 7
      else:  # if temp > 30.75
        return 8
```

Figure 7: Decision Rules for Rush Temperature Automatic Buckets

5.3.3 Rush Humidity

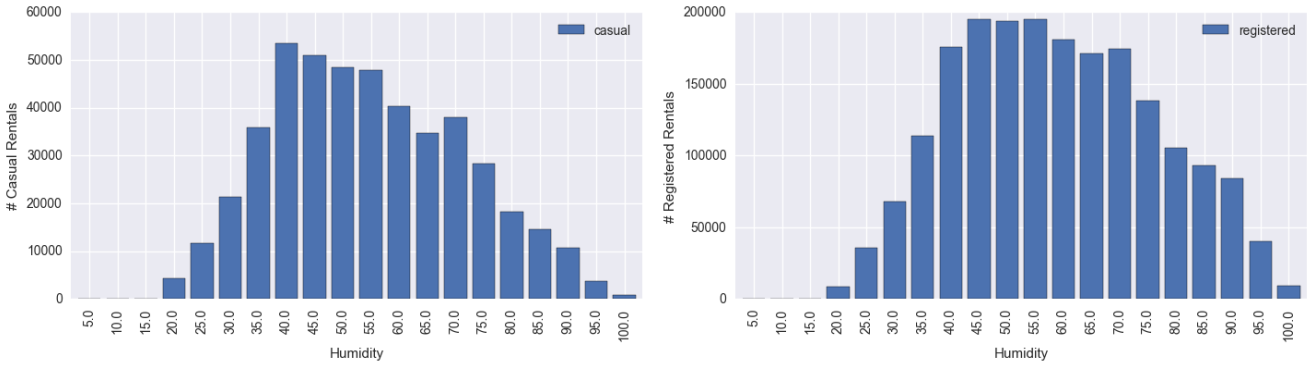

Figure 8: Histogram Humidity vs Number of Rentals

Based on the histograms of the casual and registered rentals per humidity (as seen in Figure 6), the following rules were used to assign the values of the new `rush_temp_humid_man` and `rush_humid_reg_man` features:

- Casual Rentals:
  `rush_humid_casual_man is 1 if`
  `(34 <= humidity <= 56)`
  `0 otherwise.`

- Registered Rentals:
  `rush_humid_reg_man is 1 if`
  `(39 <= humidity <= 66)`
  `0 otherwise.`

For the automatic rush humidity buckets, the respective rules that were generated by using a decision tree, with $depth = 2$, can be appreciated in Figure 9.

5.3.4 Rush Wind-speed

Based on the histograms of the casual and registered rentals per wind speed and considering how are they distributed similarly for both casual and registered rentals (as seen in Figure 10), a single rule was used to assign the values of the

```
def rush_humid_casual_bucket(humidity):
  if humidity <= 55.5:
    if humidity <= 39.5:
      return 1
    else:  # if humidity > 39.5
      return 2
  else:  # if humidity > 55.5
    if humidity <= 74.5:
      return 3
    else:  # if humidity > 74.5
      return 4
```

```
def rush_humid_reg_bucket(humidity):
  if humidity <= 66.5:
    if humidity <= 43.5:
      return 1
    else:  # if humidity > 43.5
      return 2
  else:  # if humidity > 66.5
    if humidity <= 84.5:
      return 3
    else:  # if humidity > 84.5
      return 4
```

Figure 9: Decision Rules for Rush Humidity Automatic Buckets

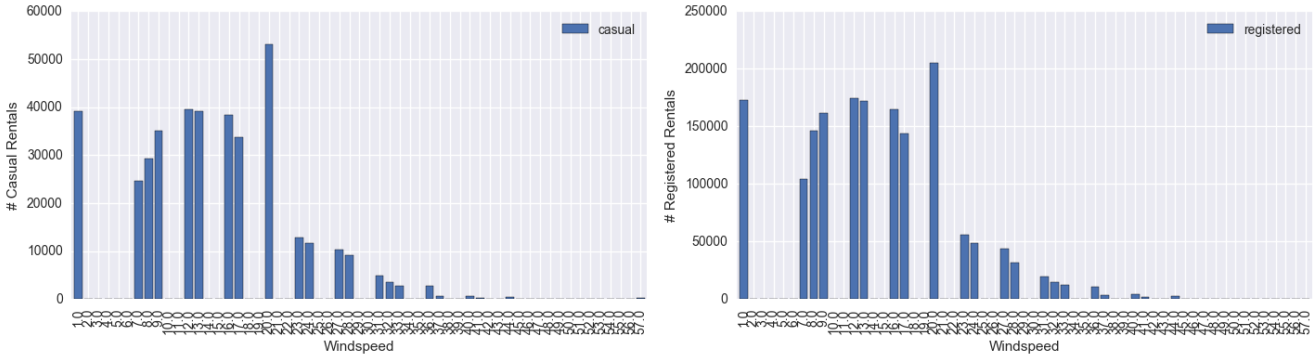

Figure 10: Histogram Wind Speed vs Number of Rentals

new `rush_wind_humid_man` and `rush_wind_reg_man` features:

- Casual and Registered Rentals:
  `rush_wind_casual|reg_man is 1 if (windspeed <= 21) 0 otherwise`.

For the automatic rush wind speed buckets, the respective rules that were generated by using a decision tree, with $depth = 2$, can be appreciated in Figure 11.

```
def rush_wind_casual_bucket(windspeed):
  if windspeed <= 9.99975013733:
    if windspeed <= 7.99980020523:
      return 1
    else:  # if windspeed > 7.99980020523
      return 2
  else:  # if windspeed > 9.99975013733
    if windspeed <= 15.9995994568:
      return 3
    else:  # if windspeed > 15.9995994568
      return 4
```

```
def rush_wind_reg_bucket(windspeed):
  if windspeed <= 9.99975013733:
    if windspeed <= 6.50234985352:
      return 1
    else:  # if windspeed > 6.50234985352
      return 2
  else:  # if windspeed > 9.99975013733
    if windspeed <= 41.9989471436:
      return 3
    else:  # if windspeed > 41.9989471436
      return 4
```

Figure 11: Decision Rules for Rush Wind Speed Automatic Buckets

6 Feature Selection

Having included new features to the dataset, now the next challenge is to identify which features should be used to predict the number of rentals. The first steep to do this is to check the correlations between the new features and the values of the casual and registered rentals.

As it can be appreciated in Figure 12, the rush rentals features (especially Rush Hours and Rush Temperature) appear to have a moderate correlation with their respective number of rentals. The environmental features have a slight negative correlation with the number of rentals (specially the rain and natural disasters indicators), which conceptually makes sense (less people bike outside if the weather conditions are not favorable). These are good indicators that these features can provide more information at the moment to predict the number of rentals.

With the previous insights, and in order to specifically select the final features to use, a set of basic features was defined as: `['humidity', 'atemp', 'temp', 'windspeed', 'weather', 'season', 'holiday', workingday', 'weekend', 'day', 'hour']`. These features are then used to generate a preliminary prediction, with its respective

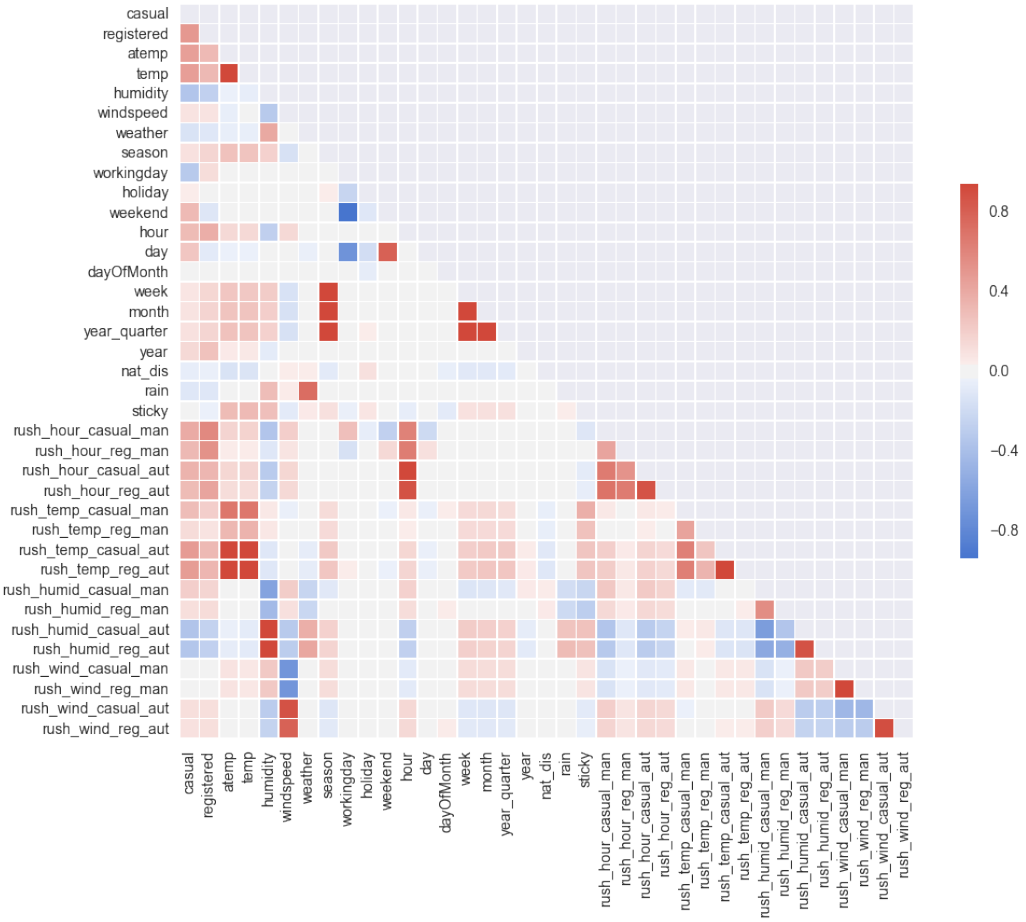

Figure 12: Complete Correlation Matrix including Engineered Features

error, using a basic Gradient Boosting Regression model (without any kind of parameter tunning). After this, iterating over each other feature to test, a new prediction is made using the basic features and the current feature to test of the iteration.

The results of this process can be seen in Table 1 where the features rush manual hours, temperature and humidity, along with the indicators for sticky weather and natural disasters, contribute to the performance of the prediction and generate a better fit for the regression. On the other hand it is important to notice that features as the month and the week overfit the model, generating worst results.

Table 1: Feature Selection Results

Features Used	Train Error	Validation Error	Test Error	Diff Against Basic
Basic Features	0.32933	0.33738	0.41050	-
With year_quarter	0.32933	0.33738	0.41050	0
With month	0.32672	0.33639	0.44116	-3.065E-2
With week	0.33768	0.34738	0.42076	-1.026E-2
With nat_dis	0.32722	0.3357	0.40773	2.75999E-3
With rain	0.32933	0.33738	0.41050	0
With sticky	0.32963	0.33764	0.41032	1.80000E-4
With rush_hour_man	0.32795	0.33777	0.40376	6.74000E-3
With rush_hour_aut	0.32933	0.33738	0.41050	0
With rush_temp_man	0.33295	0.34181	0.40820	2.3E-3
With rush_temp_aut	0.32933	0.33738	0.41050	0
With rush_humid_man	0.32940	0.33728	0.40698	3.52000E-3
With rush_humid_aut	0.32933	0.33738	0.41050	0
With rush_wind_man	0.32933	0.33738	0.41050	0
With rush_wind_aut	0.32933	0.33738	0.41050	0

Considering this, the final set of features to be used in the prediction is:

- `atemp`
- `temp`
- `humidity`
- `windspeed`
- `weather`
- `season`
- `holiday`
- `workingday`
- `weekend`
- `day`
- `hour`
- `nat_dis`
- `sticky`
- `rush_hour_casual_man` and `rush_hour_reg_man`
- `rush_temp_casual_man` and `rush_temp_reg_man`
- `rush_humid_casual_man` and `rush_humid_reg_man`

7 Prediction Models

In order to correctly predict the total number of rentals, independent models are created for both casual and registered rentals, as well as for the predictions of the year 2011 and 2012. This implies that four different models are trained and used when generating a single prediction.

With these in mind, three different regression algorithms were used to predict the number of casual and registered rentals: a decision tree regression model [4], a random forest regression model [3] and a gradient boosting regressor model [1]. These models were tunned using the features previously chosen in Section 6, and then evaluated using a 3-fold cross validation process.

The following parameters were found to be optimal for each regression model:

- Decision Tree:
  ```
  min_samples_leaf = 10
  min_samples_split = 2
  max_depth = None
  min_weight_fraction_leaf=0.0
  ```
- Random Forest:
  ```
  min_samples_leaf = 2
  min_samples_split = 10
  max_depth = 1000
  n_estimators = 1000
  min_weight_fraction_leaf=0.0
  ```
- Gradient Boosting:
  ```
  max_depth = 1000
  n_estimators = 1000
  learning_rate = 0.2
  ```

The results of a 3-fold cross validation process performed over these tunned models can be appreciated in Table 2. Considering this, the decision tree regression model is not going to be used for the final predictions. Instead only the random forest and gradient boosting regression models will be used.

With the two models selected, Random Forest and Gradient Boosting regressors, an ensemble model is proposed. This ensemble model consist in a weighted average between the results of each one of the regression models, which can contribute towards a better prediction.

Table 2: 3-Fold Cross Validation for Regression Models

Model	Error in Training	Error in Validation	Average Error
Decision Tree	0.275	0.357	0.35400
	0.278	0.347	
	0.278	0.358	
Random Forest	0.208	0.314	0.30967
	0.208	0.307	
	0.209	0.308	
Gradient Boosting	0.015	0.316	0.30933
	0.011	0.305	
	0.006	0.307	

8 Results

Having selected the two regression models for the ensemble model proposed, and using the full training with the selected features to train them, the predictions of the total number of rentals were produced and uploaded to Kaggle to verify the performance. The final results can be seen in Figure 13 and Table 3.

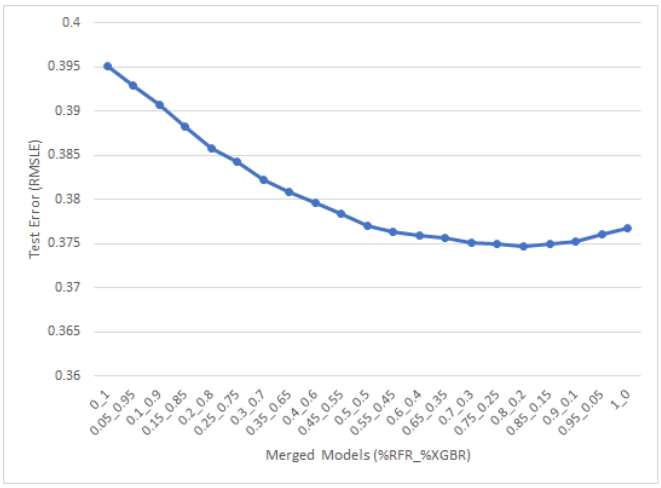

Figure 13: RMSLE Error for Ensembled Model per Weights

RFR	GBR	Test RMSLE
0	1	0.39509
0.05	0.95	0.39292
0.1	0.9	0.39065
0.15	0.85	0.38825
0.2	0.8	0.3858
0.25	0.75	0.38423
0.3	0.7	0.38229
0.35	0.65	0.38086
0.4	0.6	0.37968
0.45	0.55	0.37836
0.5	0.5	0.37705
0.55	0.45	0.37627
0.6	0.4	0.37598
0.65	0.35	0.37562
0.7	0.3	0.37507
0.75	0.25	0.37497
0.8	0.2	0.37466
0.85	0.15	0.37492
0.9	0.1	0.37527
0.95	0.05	0.37608
1	0	0.37674

Table 3: Ensemble Model Submission Results

Based on these results, the best weights for the ensemble model are: $0.8 * RF\_Prediction + 0.2 * GB\_Prediction$. With this prediction, the final RMSLE error is 0.37466, which in the Kaggle's Leaderboard ranks just below the position 92, as it can be seen in Figure 14.

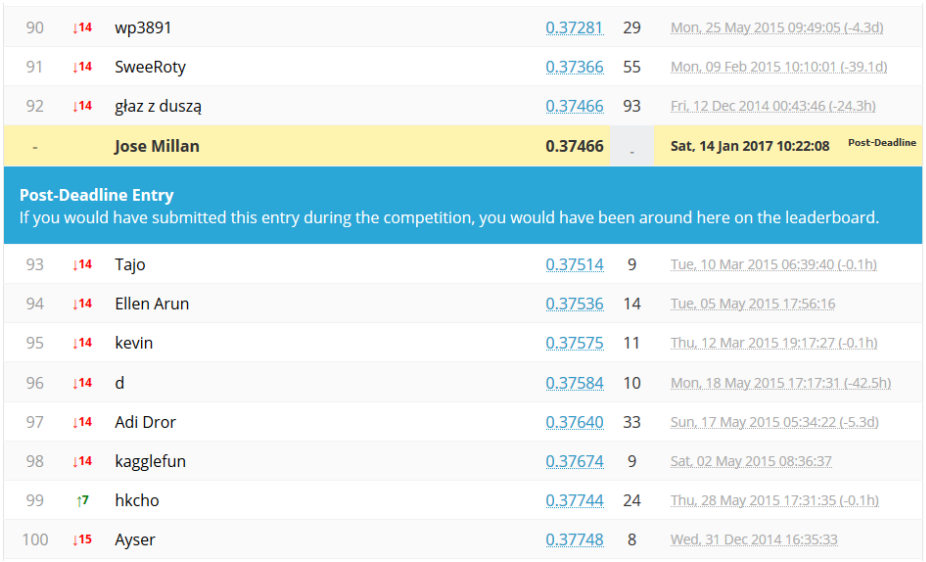

Figure 14: Kaggle's Leaderboard Final Prediction Results

9 Conclusions

It is clear that the engineered features provide more information for the regression models, allowing them to make better predictions, as is the case of the features rush manual hours, temperature and humidity, along with the indicators for sticky weather and natural disasters. On the other hand it is important to notice that features as the month and the week overfit the model, generating worst results.

With the implemented solution it was clear that the best performing regression model was Random Forest, nevertheless, an ensemble model, such as the one used in the final prediction, between two well performing models, can generate a better prediction than each model separately.

Considering the described process, the features engineered and the selected models, it can be concluded that the proposed solution provides a correct prediction for the total number of rentals of the Bike Sharing Demand Kaggle Competition, which is able to land within the Top 100 entries in the Kaggle's Leaderboard, just below position 92, with a RMSLE of 0.37466.

References

[1] DMLC. Python api reference - xgboost 0.6 documentation. `http://xgboost.readthedocs.io/en/latest/python/python_api.html`, 2017. [Online; accessed 13-01-2017].

[2] Kaggle Inc. Bike sharing demand. `https://www.kaggle.com/c/bike-sharing-demand/`, 2017. [Online; accessed 12-01-2017].

[3] scikit-learn. sklearn.ensemble.randomforestregressor - scikit-learn 0.18.1 documentation. `http://www.scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`, 2017. [Online; accessed 13-01-2017].

[4] scikit-learn. sklearn.tree.decisiontreeregressor - scikit-learn 0.18.1 documentation. `http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html`, 2017. [Online; accessed 13-01-2017].

[5] Wikipedia. List of natural disasters in the United States. `https://en.wikipedia.org/wiki/List_of_natural_disasters_in_the_United_States/`, 2017. [Online; accessed 13-01-2017].