Predicting number of Bike-share Users

A classic regression problem solved with Supervised Machine Learning

Nadir Nibras Follow
Nov 29, 2018 • 16 min read

Background

This project started as a final class-project that I worked on for a Machine Learning course (EE 660). I have thoroughly updated it as my understanding of Machine Learning (ML) concepts and practices have matured, and have transferred the code from Matlab to R in the process. The R script, the data-set and images I have used are all publicly accessible on my Github. Without further ado, let's kick into it.

Bike-share program bicycles in Washington DC

Defining the Problem and Project Goal

A **bicycle-sharing system** is a service in which users can rent/use bicycles available for shared use on a short term basis for a price or free. Currently, there are over 500 bike-sharing programs around the world. Such systems usually **aim to reduce congestion, noise, and air pollution** by providing free/affordable access to bicycles for short-distance trips in an urban area as opposed to motorized vehicles. The number of users on any given day can vary greatly for such systems. The ability to predict the number of hourly users can allow the entities (businesses/governments) that oversee these systems to manage them in a more efficient and cost-effective manner. Our **goal** is to use and optimize Machine Learning models that effectively **predict the number of ride-sharing bikes that will be used in any given 1 hour time-period,** using available information about that time/day.

Data-set used

The data-set we are using is from University of California Irvine's Machine Learning Repository. The data-set compilers used information partially from a two-year historical log corresponding to the years 2011 and 2012 from Capital Bikeshare system, Washington D.C. This information is available publicly. The compilers aggregated the data on an hourly basis and daily basis (for anyone interested in digging further). Next they extracted and added all corresponding weather and seasonal information from here.

Our data-set is a csv file (available on my Github) with information from 17,379 hours over 731 days with 16 features (information categories) for each hour. The features are:

1. *Record index*

2. *Date*
3. *Season (1:spring, 2:summer, 3:fall, 4:winter)*
4. *Year (0: 2011, 1:2012)*
5. *Month (1 to 12)*
6. Hour (0 to 23)
7. *Holiday : whether that day is holiday or not*
8. *Weekday : day of the week*
9. *Working-day : if day is neither weekend nor holiday , value is 1. Otherwise 0*
10. *Weather situation :*
    — 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
11. *Normalized temperature in Celsius. Values are divided to 41 (max)*
12. *Normalized feeling temperature in Celsius. Values are divided to 50 (max)*
13. *Normalized humidity. The values are divided to 100 (max)*
14. *Normalized wind speed. The values are divided to 67 (max)*
15. *Count of casual users*
16. *Count of registered users*
17. *Count of total rental bikes including both casual and registered*

From an initial look, the data-points far exceed the number of features, which makes this a “skinny” data-set, considered ideal for ML.

Exploratory data analysis

Before starting to process a data-set with algorithms, it's always a good idea to explore it visually. We are going to use **R** for this project. Using the **ggplot2** and **ggextra** packages, we can quickly make some plots to investigate how the bicycle usage count is affected by the features available. Now let's look at some graphs.

Scatter plot- Adjusted Temperature vs Usage

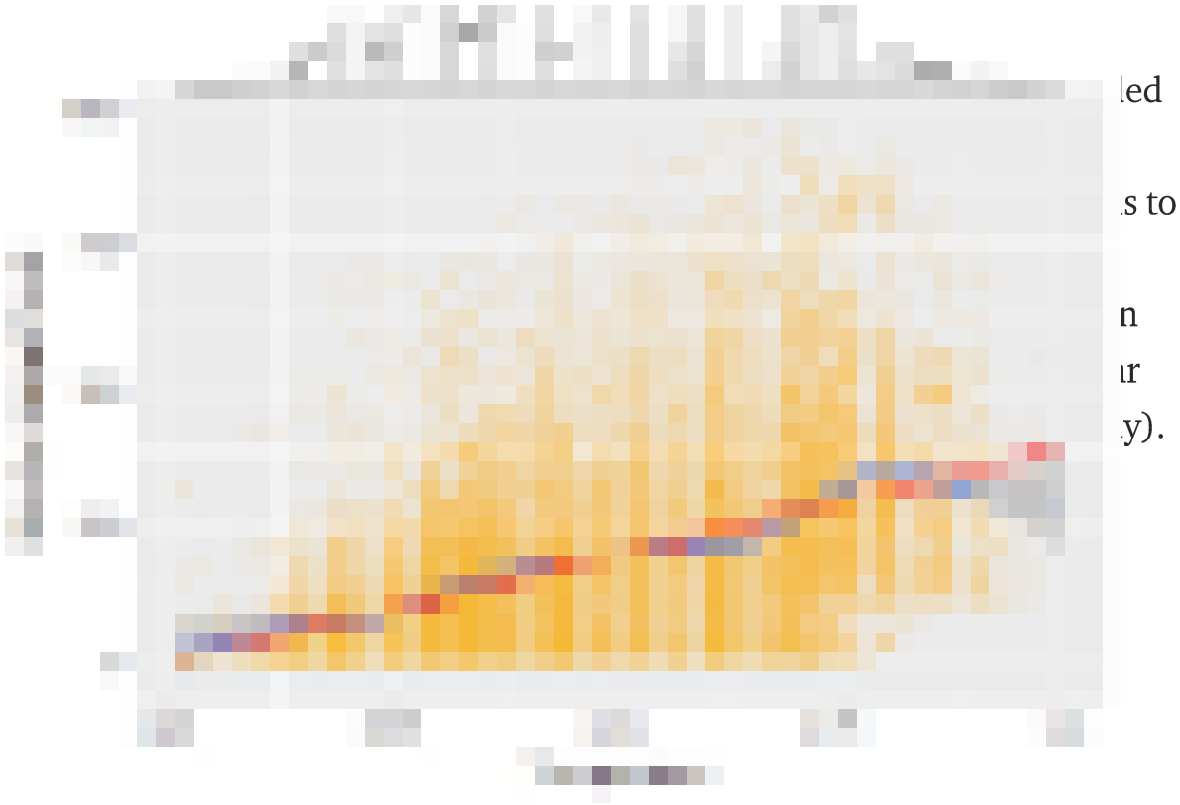
Scatter plot- Temperature vs Usage

As seen from the scatter plots above, there is a positive correlation between both temperature-to-usage and adjusted-temperature-to-usage for most of the temperature range, and a linear fit isn't far from the best-fit curve. This should intuitively make sense, as people are not likely to bike outside in cold weather. For the maximum temperatures, which seem to be a small subset of the data, there is a dip in this curve. Once again, this should make sense as users may also be discouraged to bike when it's too hot outside.

Scatter plot- Humidity vs Usage

There seems to be a negative correlation between the humidity and the usage rate, with a linear fit being very close to the best curve fit for all of the data (excluding some outliers with very low humidity). This could be explained by the

clim
corr led
pers
outc s to
hypc
dete n
exte ır
days y).

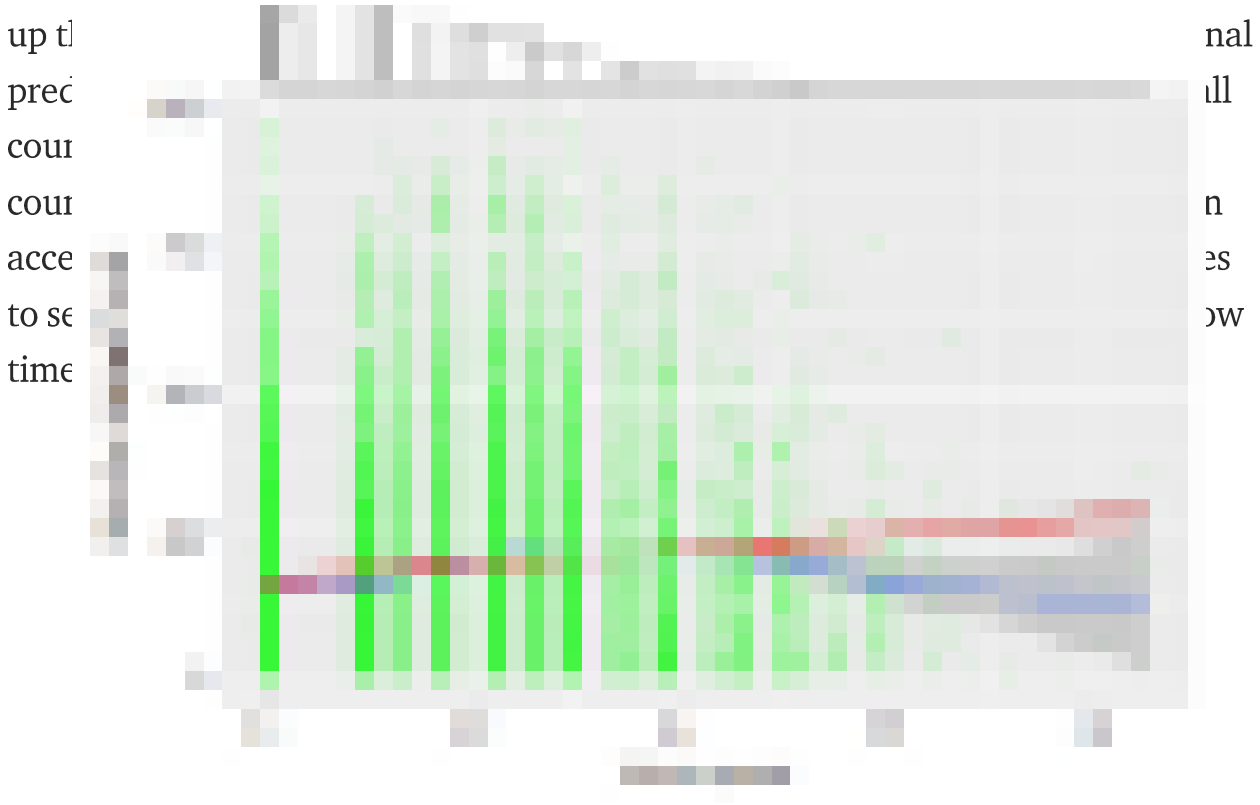


Scatter plot- Weather Situation vs Usage

Scatter plot- Wind Speed vs Usage

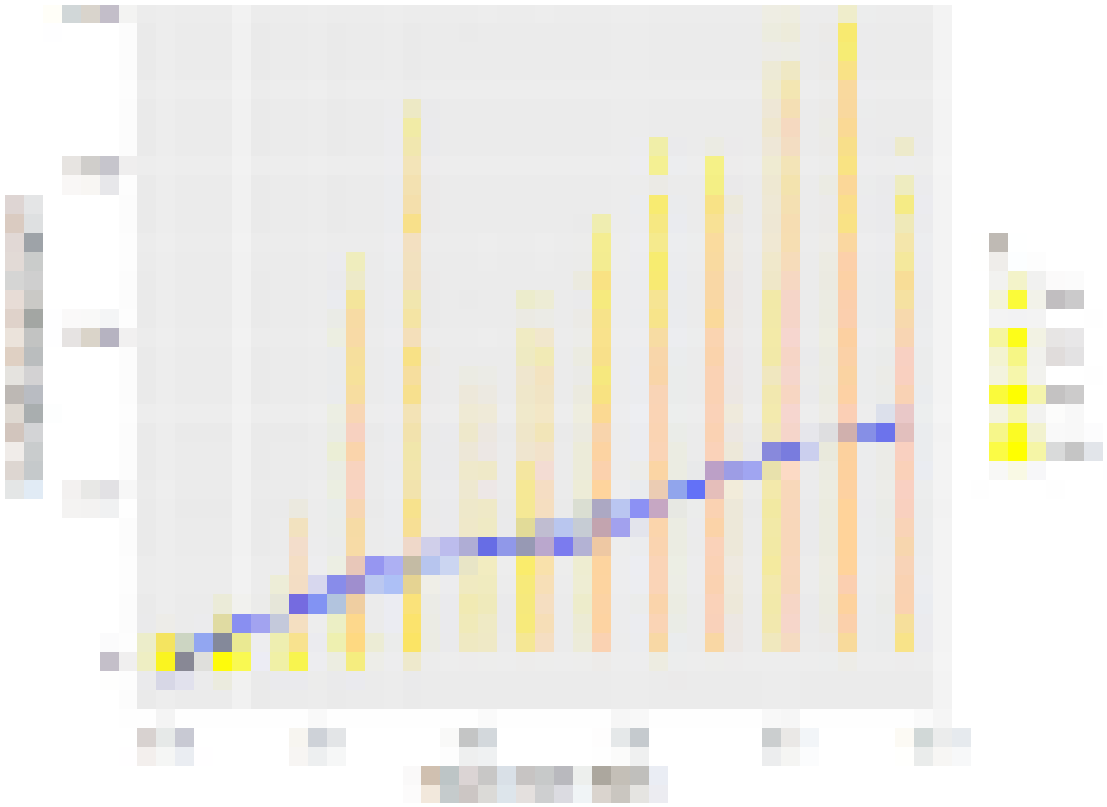
Looking at the wind speed data, however, doesn't give us a clear interpretation of how it affects usage. The correlation between the two factors is weak at best. Below is a correlation-matrix of all the continuous variables discussed so far which adds some numbers to the trends we have observed.

Note: Interestingly, the casual usage count is more correlated with the continuous variables and aligns better with our previous hypotheses. It makes sense if we think about it, as registered users who use bikes to commute to work are less likely to be deterred by uncomfortable weather conditions. We may conclude that it makes more sense to predict these two counts separately and add

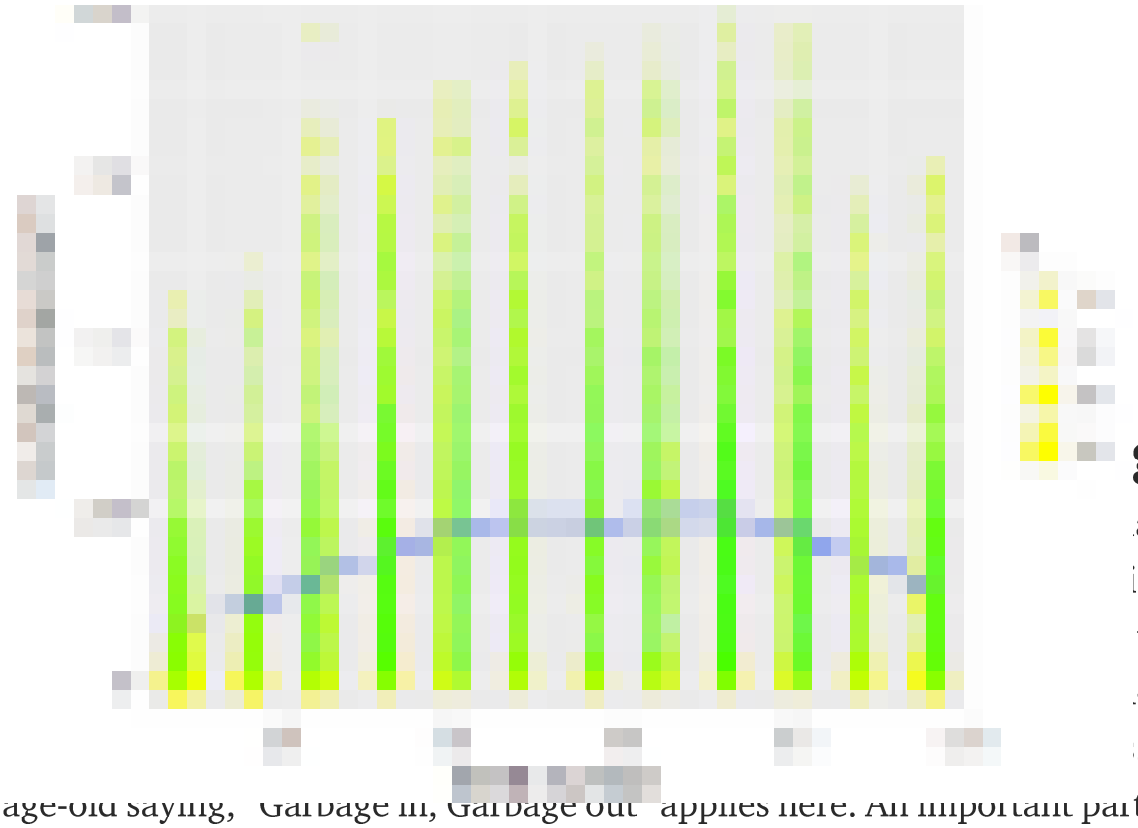


Looking at the effects of time, it seems that the lowest usage is late at night (with the minimum between 4–5 am) and the peaks are during 8–9 am and 5–7 pm, which are unsurprisingly the DC times for rush hour. The fit is far from linear. However, with some simple data manipulation (more on this in the next section), we can change this to represent the usage rate based on the temporal distance to 4 am, and find a somewhat linear fit (see below).
Note: having features that linearly predict the outcome is ideal as it reduces the need for complex non-linear ML algorithms.

A somewhat similar trend can also be observed in the month vs usage plot (below), with an evidently higher usage rate during the warmer months of the summer and the lowest usage during January. With some manipulation similar to the previous plot, this data can also used to represent usage based on the temporal distance to the month of January. This correlation, however, is not as strong as that for the manipulated time graph.



Finally, looking at the “Year” variable (below), it can be seen that the usage rate goes up from Year 1 to Year 2, which could suggest that the system grew in popularity. One important thing to note when using this variable is that, while it may be useful in the scope of making predictions in the year range provided (2011–12), the algorithm would have to extrapolate significantly to predict how it affects future dates (2018 and onward), which could make this variable a less reliable predictor for a different time period.



Pre g

For ally
an a ic
twea the
mod -
trair .

The age-old saying, "Garbage in, Garbage out" applies here. An important part of this process is Feature Engineering, which involves turning available data-features into more useful variables that can help us predict our outcome. Let's walk through the steps we are taking:

1. Using prior knowledge to remove features that don't add important information, which in this case is only the "index" feature.

2. Extracting the week number from the date. The date itself, in the format provided, isn't something that can be processed in our algorithms. From this date, however, we can extract the week number (for that particular year) and use that variable as a predictor for the usage count.

3. One hot encoding, which is the process of splitting up non-binary Categorical features (Month, Week-number, Hour, Weather-situation, Season, Weekday) into multiple binary sub-features where each sub-feature indicates whether a certain category for the original feature is True or Not (1 or 0). If possible to do so without making the data-set "fat", it is best to split each multi-classification feature up into multiple binary features. Since we have a massive data-set with 17,000+ data points, we can expand these features

4. at
are
d
gle
s
ss
s
,
eek

nts at the minimum-usage times. Therefore the temporal distances we use are calculated from 4 am for Hours, and from mid-January for the Weeks and Months.

Accounting for stochasticity of the Time Series

So far we have been dealing with the deterministic aspects of the Time Series data. For those in need of a refresher, a deterministic system is a system in which no randomness is involved in the development of future states of the system. In other words, a deterministic model always produces the same output from a given starting condition or initial state. One the other hand, if a system is stochastic, one or more parts of the system has randomness associated with it. Most Time Series data that we observe are usually best modeled with a combination of deterministic AND stochastic components. So far, we have been dealing with the deterministic aspects of our model. Now let's look at modeling the stochasticity.

Incorporating an Auto-regressive model into our features is an easy way to account for stochasticity in the system. This is based on the assumption that the

count of users at any given hour depends on the counts of users over a certain number of previous hours. This assumption is often valid in Time Series data.

***Ex**: Y(t) = B0 + B1* Y(t-1) + B2* Y(t-2)+…*

However , it's important to select the right number of lag values for our model. Autocorrelation refers to the way the observations in a time series are related to each other and is measured by a simple correlation between current observation (Yt) and the observation p periods before the current one, (Y(t-p)). The Autocorrelation Function (ACF) in R tells us the autocorrelation between current and lag values, and allows us to decide how many lag values to include in our model.

ACF results on the count per hour

Unsurprisingly, there does seem to be strong positive correlations between the user count in an hour and the previous 2 lag values, and a moderate positive correlation between the user count in an hour and the 3rd lag value. So we **add 3 lag values to our data-set as new features.**

We end up with **117 features** to predict the **hourly bike-usage** from **17,377 data points.**

Applying Machine Learning algorithms

For this project, we are going to use two of the more well-known Regression algorithms: **Maximum Likelihood Estimate** and **Maximum A Posteriori**. For linear regression, we are interested in finding the best parameters/weights, $\boldsymbol{w}$, such that given our features, $\boldsymbol{X}$, our predicted outcome, $\boldsymbol{Y\_predict = X * w}$, is as close as possible to the real outcome, $\boldsymbol{Y\_test}$.

Note on constructing training + test sets: *In most regression problems, when we are looking at cross-sectional data, we randomize the order of data-points before selecting a training and test set. However, when we are working with Time Series data, we have to carry out chronological testing since Time Series ordering is important .We cannot cut out a piece in the middle of our data as the test set, and train on data before and after this portion. We need to train on a set of data that is older than the test data.*

Packages in R and Python allow us to conveniently apply ML algorithms with two or three lines of code. While that approach has it's advantages, I think it's also important to understand the fundamental statistics and probability concepts behind our ML algorithms to understand our results better. Unlike more complex models like Neural Networks, the algorithms we will use are easier to understand and interpret. So I will very briefly go over what each of them are doing and if you want, you can go over my commented code on Github to follow exactly what we are doing step-by-step.

1) Maximum Likelihood Estimate

Maximum Likelihood Estimation (MLE) is used to estimate some variable in the setting of probability distributions. Let's say we have a Likelihood function, $P(D|w)$. This is the likelihood of having our overall data-set, D, given a certain $\mathbf{w}$ that fits the data-set features, X, to the data-set outcomes, y. When we perform MLE for w, the parameters we are trying to infer are:
$w\_MLE = argmax\, w\, P(D|w)$

In other words, we want to find the w that maximizes the likelihood, $P(D|w)$. From here, we can do a little bit of Linear Algebra, come up with a **cost/loss function** that we need in order to calculate the best weights, and then minimize this function using derivatives (recall Calculus) to find the best weights. With several underlying assumptions in place, we can find the best parameters/weights using something called the **Ordinary Least Squares (OLS)** method summarized in the formula below:

Ordinary Least Squares formula

Using OLS, we find the best parameters on a subset of our data called the **training-set**. Then we test these parameters on a different, separated subset of the data called the **test-set** in order to see how our predictions, yMLE, compare to the real outputs, yTest.

I have intentionally sped through the mathematical steps here, because: a) there are free online resources that explain each of these methods in-depth much better than I can, and b) in this article we are more focused on the applications of these algorithms than the statistics behind how they work. Note that you will require a solid understanding of Calculus, Linear Algebra and Probability Distributions to thoroughly understand either MLE or MAP.

2) Maximum A Posteriori

Along with MLE, we will also try another method called Maximum A Posteriori (MAP). MAP is derived with Bayesian Statistics, a field of statistics based on **Baye's Theorem:**

Unlike MLE which is based on Frequentist Statistics, MAP is based on a view that assumes we have some useful prior knowledge about the distributions. As the name suggests, this works on a posterior distribution, not only the Likelihood. From the formula above, we can derive the posterior distribution to be defined by the formula:

$$P(w|D) = (P(D|w)*P(w)) / P(D)$$

Assuming that $P(D)$ or the distribution of our data-set stays constant, we can conclude that:

$$P(w|D) \propto P(D|w)*P(w)$$

The first part, $P(D|w)$, is simply the Likelihood term we were working with before. And we assume that $P(w)$ follows a Gaussian distribution such that:

weights/parameters are assumed to follow a Gaussian distribution

Since we actually don't have any prior information on the weights, we don't know what **m_w** (mean) or **Tau-squared** (variance) the weight distribution should follow. So we try a thousand different combinations of the values for these terms using nested for-loops to test our algorithm on our **validation sets.** Validation sets are best defined as subsets of our training-set that are used for fine-tuning parameters before we test the parameters on the final test-set. Following the

Linear Algebra, Calculus and Probability steps we outlined for MLE, we find that the w for each training set is calculated by the formula:

After fine-tuning the parameters on the validation sets, we use these parameters on the test-set features to predict the MAP outcomes and compare these predictions to the yTest values.

Evaluating Outcomes and Model(s)' Performance

Scatter plot of MLE predictions for y VS real yTest values

Scatter plot of MAP predictions for y VS real yTest values

Summary statistics for evaluation:

MLE

Run-time: 0.70 seconds
Median prediction error: 27.05
Mean prediction error: 56.53
R-squared value : 0.65

MAP

Run-time: 258.36 seconds
Median prediction error: 27.93
Mean prediction error: 57.85
R-squared value : 0.63

A useful approach is to compare the performance of our models against the naive forecast. Naive Forecast is an estimating technique in which the last period's values are used as this period's forecast, without any adjustments that attempt to establish causality.

Naive Forecast model

Median prediction error: 37
Mean prediction error: 69.9
R-squared value : 0.65

Examining how MLE, MAP and Naive Forecast predictions vary from the real data using a sub-set of the Test data

Discussion

As can be seen from the plots and the values above, MAP and MLE give us very similar results, as is expected when we use MAP without any prior information about the target function we are trying to derive. We find that we have median prediction errors of around 27 and mean error predictions of around 57. This is significantly better than the naive forecast model with a median prediction error of 37 and mean prediction error of 69.9.Note that the overall range of user counts is from 0 to almost 800.

The R-squared value for both models and the naive forecast was approximately 0.63–0.65. So our models do not perform better than the naive forecast when looking at this metric. The best fit lines through the scatter plots are very close to what would be perfect, a line with gradient of 1 passing through the origin. Note that none of our predictions are less than zero because we assumed that the usage count follows a Poisson distribution and we used a Poisson Regression model.

Investigating a little bit further, let's look at a subset of our test-data which has just 100 data-points. We see that both MAP and MLE predict the user-count very well when the overall usage count is low, but not so well for the instances when the usage count was over 450. These points lead to outliers with large errors which explains why our mean errors are so much larger than our median errors.

Considering how much much lower our median and mean errors are than the naive forecast, it is probably safe for us to conclude that we have created effective models for predicting the number of bike-share users per hour. There is certainly room for further improvement, especially for hours when the number of users is high. So if you have suggestions on what could be done to improve these models, please feel free to let me know. That concludes our work here. Thank you for reading.

If you made it this far, I hope you enjoyed reading this as much as I enjoyed writing it. I also hope you know more about Exploratory Data Analysis and Machine Learning than you did before. If you think this may me interesting or educational to others you know, please do share it with them. If you liked the article, and had something you wanted to share with me, feel free to comment or contact me via email at nadir.nibras@gmail.com or at https://www.linkedin.com/in/nadirnibras/.
I am committed to improving my methods, analyses or data-sets, so if you have any suggestions, please do let me know. If you want to follow more of my work on data science, follow me on Medium and Linkedin.
If you are a fan of Data Science or it's applications in understanding the world, please do connect with me. It's always fun talking to fellow stats nerds and I would love to collaborate on projects :)