# The Divvylorean - A Time Machine for Future Divvy Station Status - Dean Marano

## Introduction

In 2013, Chicago installed a public bike system called Divvy Bikes. Within a year, it has over 23,000 members (http://divvybikes.tumblr.com/post/96460660270/new-divvy-data-now-available) taking an average of 13,000 rides daily. It's quickly become another standard means of transportation in Chicago for many people, and has the added benefits of being extremely environmentally friendly, healthy, and reducing the necessary number of vehicles on the road.

I joined Divvy less than a month after it opened, and have taken over 450 rides within the past year. While I love Divvy, I frequently run into the problem of the bike dock near work filling up before I get there, or not having a bike when I go to leave. Both of these means me walking to the next closest station, which is undesiredable. During the summer, I realized if I left 15 minutes earlier, I could normally have a spot to park my bike. I was onto something.

In November of 2013, I noticed that Divvy's public API had current station data - knowing how many bikes were available, docks were available, and some other information for every station. Not yet knowing what I would do with it, I started collecting this data, once every 15 minutes, every day, for over a year. Fast forward to now, and over the last few weeks I've learned some different skills for dealing with data predictions, and decided to put the data to good use.

My goal is to be able to predict whether or not there will be bikes or docks at a station depending on the day of the week, time of day, temperature, and weather. I have the data I've collected from Divvy, as well as data from the Forecast.io API, which provides hourly historical data. By combining these, I had station and weather data for over 22,000 data points.

## References

When deciding on my input values, I came across a paper describing the issues with correlated features in Bayesian classifiers (Langley & Sage, 1994). Originally, I was going to include more detailed weather data (humidity, pressure, etc) but decided against it in order to have less chance of having overlapping features.

When developing the strategies I would be using, I read up on different Bayesian strategies, specifically around dealing with continuous inputs and outputs. I found some articles relating to dealing with continuous outputs, however one of the things I was looking for was to have as quick and efficient predictions as possible, allowing me to generate them and give them to users instantaneously. One solution, called Flexible Bayes (John & Langley, 1995), improved upon the Gaussian output, however at the cost of memory and speed issues. There were various strategies for dealing with continuous input. The first approach is bucketing, which is relatively straightforward, with the downside that it isn't as precise. I found some information more information about doing coninous function inputs in an unpublished chapter of the Machine Learning textbook (Mitchell, 2005) which also provided some insights.

## Methods

To collect the data originally, I was simply using curl to pull the Divvy data, triggered by cron every 15 minutes. This resulted in each point in time being represented by an individual file. While this was the easiest way of fetching data, dealing with 20,000+ individual files is not something many tools handle easily, and I ran into open file limits when originally dealing with the data. Instead of having very many small (100K text) files, I switched to grouping the data by station. Since there are only 300, this was a much easier number of files to deal with, although it meant dealing with larger data files (3MB text minimum).

I combined these files using the combine-into-stations.js script, a Node.JS script. After combining the data points, I removed variables that didn't change between points (such as names) to reduce space needed. Next, I combined the weather data for each point into all of the station files. This is the addWeatherData function from the read-station.js file. The last step in the file processing is doing counts for each state, for use in calculating the P values.

The first issue I ran into was figuring out how to represent time. Since I had 96 different times, over each day of the week, first I split the time information into a weekday and a time. Then, I decided to simply break each time (12:00, 12:15, etc) into it's own category, represented in minutes since midnight. This allows the data to more closely follow the cyclical nature of the week and day cycle, while also making it easier to process. For a larger data set, I think incorporating the week of the year, day of the month, month, and day of the year in might be helpful (especially for finding irregularities on holidays, or cyclical patterns around the beginning and end of the month). I didn't expect to see any of those in the data that I was collecting, so I stuck to just using day of week and time of day.

After doing the counts on the backend, the files are ready for presentation and calculation from the web app. When the page is loaded, the site loads the correct file for the station requested, and, using the current day and time, combined with the current weather data, presents an estimation to the user.

The initial implementation was simply a boolean for whether the station would have bikes or have docks. While useful, I thought it would be more interesting to try and predict how many bikes and docks would be available. Unfortunately, Bayseian outputs are binary and not continuous. However, due to the limited scope of the output (normally between 0 and 40) I decided that it would be fairly straight forward to compute, for each station, the odds of a certain number of bikes being at every station. That is, compute the odds of there being 1 bike, 2 bikes, etc. Then, by taking the maximum of the odds, I can give a guess as to how many bikes should be expected. This allows the simplicity of the Bayes approach with a semi-continuous output. Also, by graphing these numbers gives the user a good idea of other possible outcomes and their likelihood.

My reasoning for doing this in the browser is I wanted to give the most immediate feedback to the user as possible. By pushing the data analysis as close to the user as possible, I'm able to give the most accurate results possible. Whenever the page is loaded, the current weather data is fetch, and the current time of day and date information is used. I used Javascript web framework, Ember.js, for doing the data binding and fetching. I have a lot of experience with this so it was the easiest course of action for me.

## Results

The results I had were encouraging. From the stations I was working with, the results I were seeing were as expected, to be around the 90s. As times grew closer to rush hour, the likelihoods decreased. When the weather was warmer, the less likely you were to get a bike. Rainy weather made it more likely to have a bike.While my primary prediction wasn't always correct, the actual number was normally within the top 5 guesses.

Unfortunately, when spreading the data to predict the actual number of bikes, holes in the data appeared, where the predictions dropped suddenly to zero. This, for me, goes to show that the more data you have, the more meaningful your results will be.

## Discussion

I think what I did fairly well addressed the prediction problem I was looking to solve. In doing so, I think one of the most valuable learnings was figuring out how to transform data to be categorical or continuous, and what the implications of those conversions were. By grouping date and time the way I did, I get repetative data very well, although that may not work well in a continuous system environment. Also, any information about the previous data point is not used for the prediction of the current data point, so there is some information loss from that aspect. One change that could be made is instead of looking at the counts, look at the deltas (that is, bikes in and bikes out) and give predictions using that information plus the current data. While this wouldn't be able to give as meaningful long term data (that is, to say on days when it's 60 it's unlikely to find a bike at 5PM) but it would definitely improve the accuracy of near term predictions.

While Naive Bayes was not the most intensive implementation, I think the results were acceptable, given the fairly straight forward data, consistent data inputs. I didn't have to deal with any incomplete data points which was helpful, but I think a large number of problems can be solved with fairly simple algorithms. Ithink a neural network would be able to handle the input data fairly well also, and it may be interesting to try that and compare the results.

## Next Steps

I'd like to pull more information, such as showing the current station status, when giving my predictions. Also, being able to manipulate when the prediction is for (day of week, weather conditions, time) via sliders would give interesting insight into how each factor plays into the prediction. Also, I would like to use the current station status to get a gauge of how accurate the predictions are, and collect data about whether the predictions are acurrate or not.

It'd be neat to also develop the user experience more, and be able to show two stations (a departure point and destination), showing how many bikes are available at departure and how many docks are available at the destination. It'd be pretty easy to calculate the duration of time from each station to station using Google Maps' API, and add that travel time into the prediction so the destination prediction would be accurate for when the person is expecting to arrive.

I plan on continuing work on this project after this class, refining the experience and exploring different machine learning techniques and their effectiveness.

## References

John, George H., and Pat Langley. "Estimating continuous distributions in Bayesian classifiers." Proceedings of the Eleventh conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1995.

Langley, Pat, and Stephanie Sage. "Induction of selective Bayesian classifiers." Proceedings of the Tenth international conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1994.

Mitchell, Tom M. Machine Learning. New York: McGraw-Hill, 2005 (http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf).