Incentivizing Riders to Redistribute Bikes

*A model for predicting empty bike station events N hours before they happen*

**Order of Scripts:**

1. ETL\_00
2. ETL\_01
3. ETL\_02
4. predict\_timeseries
5. predict\_logisticreg
6. predict\_decisiontree
7. predict\_randomforest
8. remove\_consecutive\_positive\_observations

**Introduction:**

Bike share companies rely on customers annually subscribing to their service for bike access. To avoid churn these companies must consider everything that effects customer satisfaction. One of the most important aspects of keeping customers happy in a through a bike share program is providing them with reliable access to bikes and places to park bikes. If a customer arrives at a bike station ready to take a bike, but finds no available bikes at that station it can be a huge interruption in their daily routine. A top priority for bike shares needs to be minimizing the times when a station is completely empty of full of bikes.

The goal of this analysis and model is to predict when individual stations will become empty a few hours before of the ‘empty event’. If it is possible to accurately predict empty events than bike shares can either hire employees to redistribute bikes to these empty stations, or incentivize subscribers through there app to redistribute bikes to obtain free rides.

**Obtaining and preparing the data:**

The greatest challenge in predicting empty events is preparing the data. Bike share companies openly share data from their systems, but the data that they share is trip oriented. A single row of data is a trip with a start and end date/time as well as a to and from station. The first step is to transform the data into start and end occupancy for each station.

Trips data as provided:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Trip id | Start time | Stop time | From Station | To Station |
| 1 | 10:00am | 10:05am | A | B |
| 2 | 11:00am | 11:10am | B | C |

We can visually see from the above table that the bike arrived at station B at 10:05am and left station B at 11:00am. Using SQL logic each consecutive trip can be lined up to create a row occupancy at stations. This workflow is completed in the ETL\_01 file.

Occupancy data after ETL:

|  |  |  |  |
| --- | --- | --- | --- |
| Occupation id | Arrival time | Departure time | Station |
| 1 | 10:05am | 11:00am | B |

*Example Syntax (converted to rank and join in pandas)*

SELECT

RANK( PARTITION BY bike\_id ORDER BY trip\_id ) AS ‘rank’,

Stop\_time AS ‘arrival\_time’,

To\_station AS ‘station’

FROM

‘trip\_data’

LEFT JOIN (

SELECT

RANK( PARTITION BY bike\_id ORDER BY trip\_id ) - 1 AS ‘rank’,

Stop\_time AS ‘arrival\_time’,

To\_station AS ‘station’

FROM

‘trip\_data’ ) offset

ON

Offset.rank = rank

After the trip data has been transformed into occupancy data it can be used to count the number of bikes occupying a station at any given minute. For each 530 stations the number of bikes need to be counted for every minute that the station was open. This can be done by evaluating every row of occupancy of data and adding a +1 for every minute between the arrival and departure to a dictionary of minutes for the station the occupancy occurred at.

|  |  |  |  |
| --- | --- | --- | --- |
| Occupation id | Arrival time | Departure time | Station |
| 1 | 10:05am | 11:00am | B |

Occupancy converted into counts per station, per minute. This workflow is found in the ETL\_02 file. It saves a csv file for each station.

|  |  |  |
| --- | --- | --- |
| Station | Time | Count |
| B | 10:05am | 1 |
| B | 10:06am | 1 |
| B | 10:07am | 1 |
| B | 10:08am | 1 |
| B | 10:09am | 1 |

This is an extremely time consuming process to run. There are approximately nine million rows of data in the occupancy table and the average occupancy is 600 minutes. Every row needs to be pivoted as shown in the example above. This resulted in 5.2 billion iterations to transform the data into occupancy counts.

I used a few different techniques get the script time per station down to approximately 1.5 minutes so that the 535 stations could run over 13 hours ((535 x 1.5) / 60). Occupancy counts only needed to be stored per station at 30 minute increments so memory was not the issue a major issue. Because of this I was also able to design the script in a way that it could be spread across processes. Chunks of 4 stations can be run in unison bringing run time closer to 3 hours. The final output is a folder with a single csv for every station. Each row of data is a thirty-minute period displaying the mean, min and max number of bikes occupying that station during the time period. This format made it easy to test different models out on stations without having to load the data for every station into memory.

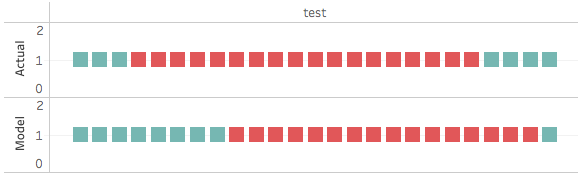
**Model + Analysis:**

In order to redistribute bikes to stations before empty events there needs to be a reasonable amount of provided to make the move. I focused on the scenario in which customers would be incentivized to move bikes via free rides. In this case we would likely want to incentivize them a few hours ahead of the event. To do this I chose to use the occupancy counts from 2 hours prior to the event at question.

I worked through three classification techniques to evaluate the probability of a given 30-minute period of time containing an empty event. Evaluating these models turned out to be more difficult than expected because the number of empty events rarely made up more than a few percentages of the overall data set. Testing the accuracy of a predicting all 0’s produces scores of 95%+. I chose to use the True Positive rate to determine the accuracy of my models (TP / (FP + TP)).

A variety of combinations of features were used for each model from just the mean occupancy count from 2 hours to prior to hours, months, and the mean occupancy counts for up to 8 hours prior to the event. The models consistently weighted the mean occupancy counts from 2 hours prior as the most important feature.

Using just the prior 2-hour mean occupancy counts in a decision tree with max depth of 3 I was able to produce an average True Positive rate of 60% across all stations. This seemed unusual and a closer look at the data showed why. Each mark is an observation. Red marks are empty station events (1). The top row are the actual observed values while the bottom row is the model predictions. At first it looks like the model is doing a fairly good job spotting empty station events, but in reality it is simply a reactionary model. It does not start predicting empty events until the empty event has already begun.



In an attempt to overcome this I tried transforming the data to only mark the first event in a series as positive observation. I did this using a for loop to look for repetition in positive observations and isolate the first one in any series:

previous = 0

list = []

for i in data.minutes\_empty:

current = i + previous \* i

previous = i

list.append(current)

After a few quick attempts at modeling this transformed data I found that there was very little I could find in the form of features that was predictive of a positive observation. On top of that the number of positives observations were event further limited by removing consecutive positives.

**Conclusion:**

The challenge of predicting empty station events in bike share systems is an extremely interesting one. The fact that it is a closed system (finite amount of stations with a finite amount of bikes) makes me think that there is a possible solution to the problem. I believe the solution will require a very complex model that effectively looks at the state of the entire system (all stations) to predict the likelihood of a station becoming empty.