Code

Exploratory Analysis of Candidate Features for Bottleneck Identification

The 5-Minute Station Data available on the CalTrans PeMS Data Clearinghouse has the following field specifications:

- Timestamp
- Unique Station Identifier
- District #
- Freeway
- Direction of Travel (N | S | E | W)
- Lane Type
  - CD (Coll/Dist)
  - CH (Conventional Highway)
  - FF (Fwy-Fwy connector)
  - FR (Off Ramp)
  - HV (HOV)
  - ML (Mainline)
  - OR (On Ramp)
- Length (of segment covered by station)
- Samples received from all lanes during this time interval
- Total Flow (Veh/5min)
- Avg Occupancy (%)
- Avg Speed (Mph)

PeMS uses the algorithm described in, "Systematic Identification of Freeway Bottlenecks" (Chen et al.) to identify bottlenecks on California freeways. It relies on the calculation and thresholding of speed differentials between one Vehicle Detector Station (VDS) to another.

In this document, we will explore the 5-Minute Station Data and evaluate which signals captured by VDS systems are appropriate for identifying bottlenecks. The final goal is to categorize the bottlenecks (for example, distinguishing between recurrent and non-recurrent events) so that traffic solutions can be made with higher quality analysis.

Most terms/vocabulary used in this analysis are defined in the CalTrans PeMS Glossary.

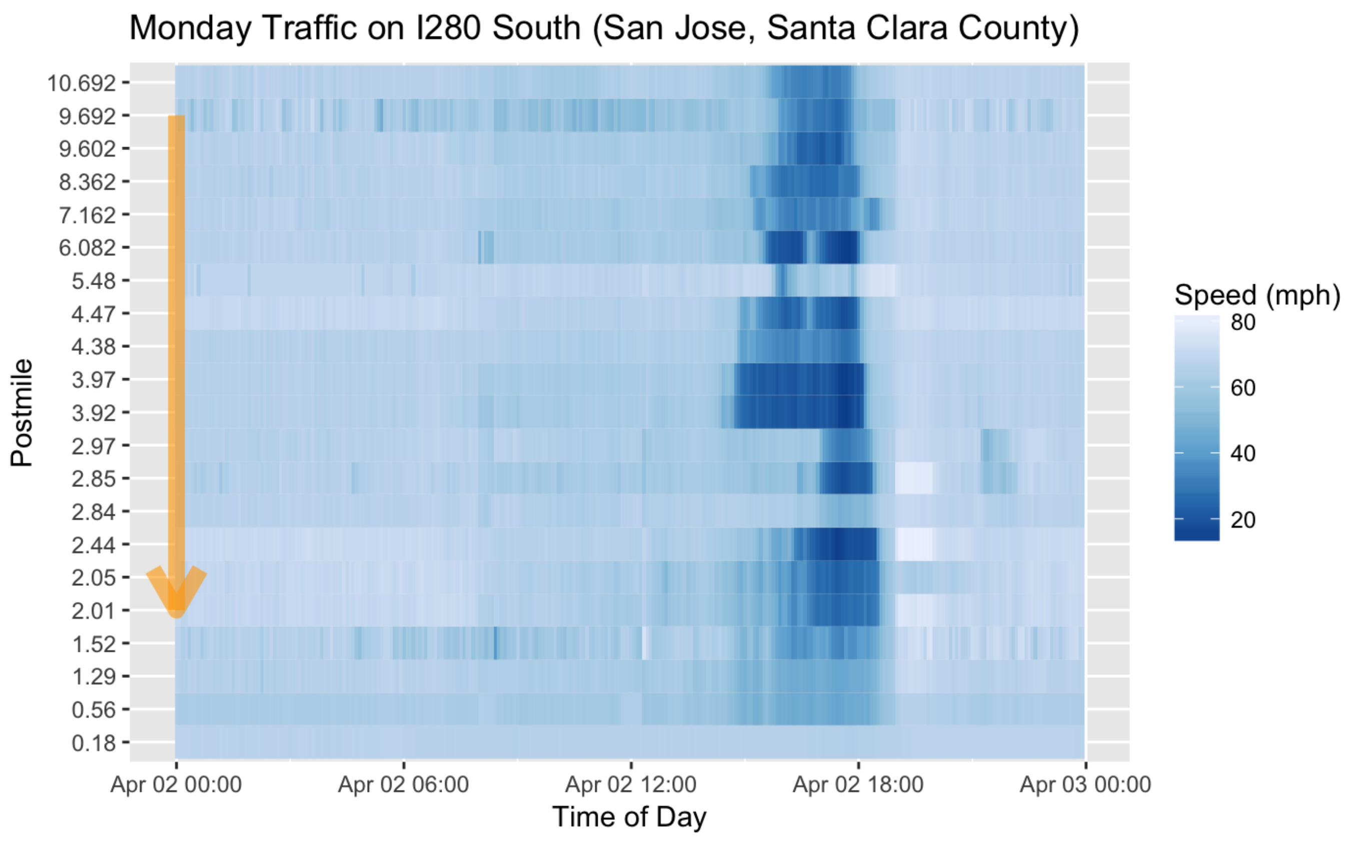
Notable Notes worth Noting:

- Only weekday traffic is considered.
- We do not use lane-specific data (via loop detectors); the dataset has been subsetted to only the average metrics for each VDS at each 5-minute time increment.
- The original paper does not account for any interaction of ramps (on/off) and mainline (ML) segments of the highway. It is likely that the algorithm is used indiscriminantly, but we focus our analysis on stations of `type=ML`.

Part I: Implementing the PeMS Bottleneck Detection Algorithm.

First, we wish to replicate the results in the original bottleneck identification paper. Assuming that their algorithm reliably detects bottlenecks, it would more practical to build our own version with potential for modifications, rather than having to download their list containing a limited set of available metrics.

To test our implementaiton, we selected the strip of mainline highway on I280S in San Jose on 04/02/2018. The detectors lead directly into central/downtown San Jose, and as shown by the 2D plot below, heavy traffic congestion occurs in the mid-late afternoon.



To detect bottlenecks, we follow the four inequalities presented in the paper. x_i is upstream of x_j, represented by $x_i < x_j$.

- $x_j - x_i < 2mi$
- $v(x_k, t) - v(x_l, t) > 0$ if $x_i \le x_k < x_l < x_j$
- $v(x_j, t) - v(x_i, t) > 20mph$
- $v(x_i, t) < 40mph$

In plain English, they mean:

- A bottleneck is defined as a slowdown over a short segment (less than 2mi).
- Bottlenecks can be calcuated using two non-adjacent detectors, but only if the speed is continuously rising.
  - The formula doesn't actually make sense, and I assume it is an error in the paper. If $x_i \le x_k < x_l < x_j$, then $v(x_k, t)$ should be less than $v(x_l, t)$ in order to express a continuous rise in speed.
- A bottleneck ends (downstream) when the congestion improves by at least 20mph.
- A bottleneck is only recognized (upstream) at speeds lower than 40mph.

Using these conditions, we wrote a couple functions that take VDS data and return the most downstream location of each active bottleneck.

Hide

```
search_slowdowns = function(i,df,maxD=2,mph_trigger=20,direction=T){
  # Given the VDS and timestamp at row (i) of Dataframe (df),
  # check surrounding VDS signals for evidence of slowdowns exceeding the value of
(mph_trigger)
  # Args: (see parameters of bottleneck_finder() function)

  # Returns:
    # candidates (vector) - a named vector of VDS identifiers that satisfy the ineq
ualities I-IV.

  candidates_vec = vector()
  dir_int = ifelse(direction,1,-1)

  row_i = df[i,]     # starting time,point,speed
  df = df[which(df$timestamp == row_i$timestamp),]      # time is constant
  df$rel_PM = df$abs_PM - row_i$abs_PM                  # get distances relative to
given VDS (can be negative)
  df = df[order(df$abs_PM),]                            # sort detectors (increasin
g postmile)
  i = which(df$rel_PM == 0)

  # check downstream:
  down = dir_int
  while (1 <= i+down & i+down <= nrow(df)){
    row_j = df[i+down,]
    if (abs(row_j$rel_PM) > maxD) { break }    # Inequality I
    else if (df[i+down-dir_int,]$speed - row_j$speed > 0 &&
        row_j$speed - row_i$speed > mph_trigger){
        # report candidate bottleneck
        candidates_vec = c(candidates_vec,row_j$station)
    }
    down = down + dir_int
  }

  # check upstream:
  # up = dir_int
  # while (0 < i+up & i+up <= nrow(df)){ # Inequality 2
  #   row_j = df[i+up,]
  #   # Inequality 1 and 3:
  #   if ((row_j$speed - row_i$speed > mph_trigger) && (abs(row_j$rel_PM) < maxD)){
  #     # report candidate bottleneck
  #     candidates_vec = c(candidates_vec,row_i$station)
  #   }
  #   up = up + dir_int
  # }
  candidates_vec = unique(candidates_vec)
  names(candidates_vec) = rep(row_i$timestamp,length(candidates_vec))
  if (length(candidates_vec) != 0) { return(candidates_vec) }
}
```

Hide

```
bottleneck_finder = function(df,minSpeed=40,maxD=2,mph_trigger=20,direction=T){
  # Identify bottlenecks using the algorithm described in Chen et al.

  # Args:
    # df (DataFrame) - contains the following fields:
        # timestamp (converted using the as.POSIXct() function)
        # VDS Identifier
        # abs_PM (absolute post mile)
        # speed (mph)
    # minSpeed (mph) - Inequality IV. No bottlenecks exist until highway speeds (up
stream) drop below this value.
    # maxD (mi) - Inequality I. Represents the maximum separation of upstream/downs
tream detectors (which can be consecutive or non-consecutive)
    # mph_trigger (mph) - Inequality 3. A bottleneck ceases when congestion improve
s by this amount.
    # direction (bool) - F if traffic flows in the opposite direction of post mile.

  # Returns:
    # bottlenecks (DataFrame) - a subset of (df) with only rows identified as bottl
enecks by the PeMS algorithm.

  # Inequality IV:
  sub40_index = which(df$speed < minSpeed)

  # Searches both upstream and downstream of each detector every 5 minutes:
  test_candidates = sapply(sub40_index, search_slowdowns, maxD = maxD, mph_trigger
= mph_trigger, df=df, direction=direction)
  test_candidates = unlist(invisible(test_candidates))
  bottlenecks = data_frame(station = test_candidates, timestamp = names(test_candi
dates)) %>% group_by(timestamp) %>% slice(1) %>% ungroup()

  bottlenecks$timestamp = as.POSIXct(bottlenecks$timestamp, format="%Y-%m-%d %H:%M:
%S")
  bottlenecks = left_join(bottlenecks,df)
  return(bottlenecks)
}
```

The bottleneck locations detected by this code are tabulated below, along with the number of times they occur.

```
In order of Post Mile:
0.56 2.44 2.85
  14    4   23

In order of VDS:
401809 401943 403409
    14     23      4
```

We compare these results to query results from the “Top Bottlenecks” page on PeMS:

Location									Bottleneck Characteristics			
VDS	Name	Type	Shift	Fwy	Abs PM	CA PM	Latitude	Longitude	# Days Active	Avg Extent (Miles)	Avg Delay (veh-hrs)	Avg Duration (mins)
403406	S 1st St rm-s-loop	ML	PM	I280-S	2.010	2.01	37.32592	-121.883314	1	0.9	155.9	65.0
401943	Bird Ave	ML	PM	I280-S	2.850	R2.85	37.322361	-121.8978	1	2.1	349.6	85.0
400429	LAWRENCE EXPWY	ML	PM	I280-S	8.362	6.98	37.319424	-121.993175	1	1.2	31.1	15.0
400292	Saratoga ave rm-s-diag	ML	PM	I280-S	7.162	5.78	37.316378	-121.972052	1	4.6	125.7	15.0
401388	BASCOM & LELAND	ML	PM	I280-S	6.082	4.7	37.316628	-121.952438	1	5.5	952.4	100.0
403414	Southwest Expwy/Meridian Ave	ML	NOON	I280-S	3.920	3.92	37.314844	-121.913648	1	0.7	19.5	15.0
401811	N OF 1ST STREET	ML	PM	I280-S	2.050	R2.05	37.325651	-121.883953	1	0.6	5.6	5.0
403414	Southwest Expwy/Meridian Ave	ML	PM	I280-S	3.920	3.92	37.314844	-121.913648	1	1.5	582.5	130.0
403404	S 7th St rm-s-diag	ML	PM	I280-S	1.520	1.52	37.327807	-121.874899	1	1.2	225.3	85.0

It is clear that our results do not match their results. Here are some suspected reasons:

- The results from the PeMS site do not seem to enforce the 2-mile maximum (Inequality #1 of the algorithm), as the **Avg Extent** column has entries up to 5.5 miles in length.
- In general, our results are not reporting bottlenecks that exist upstream. It is possible that the other parameters, such as the necessary speed differential (default 20mph).
- PeMS runs their algorithm in 3 separate blocks in the day (AM: 5-10, NOON: 10-3, PM: 3-8), which can over-report bottlenecks that persist at the transition point (i.e. 10AM or 3PM). However, the difference would be noticeable and relatively inconsequential.
- The authors of the original paper explain that only *sustained bottlenecks* are of interest, for which they provide a formula indicating that a bottleneck must persist for at least 25 minutes out of a 35 minute period. However, both the website and our implementation ignore this condition, with bottlenecks that last as little as 5 minutes being reported.
- Finally, our interpretation of the algorithm may be incorrect. The original paper does not go into detail about how their algorithm is implemented, so we were forced to make a some assumptions.
    - For example, our implementation originally scanned both upstream and downstream, and the results were closer to those displayed in the PeMS table, despite the method being in violation of Inequality IV.

We re-run the code with slightly more relaxed parameters:

Hide

```
detected_bottleencks_v2 = bottleneck_finder(speed_280S, maxD = 5.6, mph_trigger = 1
0, minSpeed = 45, direction = F)
```

```
Joining, by = c("station", "timestamp")
```

```
In order of Post Mile:
 0.56   2.44   2.85   4.47 6.082
   12      4     20      7     1

In order of VDS:
401163 401388 401809 401943 403409
     7      1     12     20      4
```

Even with the adjustments, the results still do not resemble the bottlenecks reported by PeMS. Therefore, we will have to proceed without a code implementation of the bottleneck identification algorithm.

Part II: Clustering with PeMS-provided Bottleneck

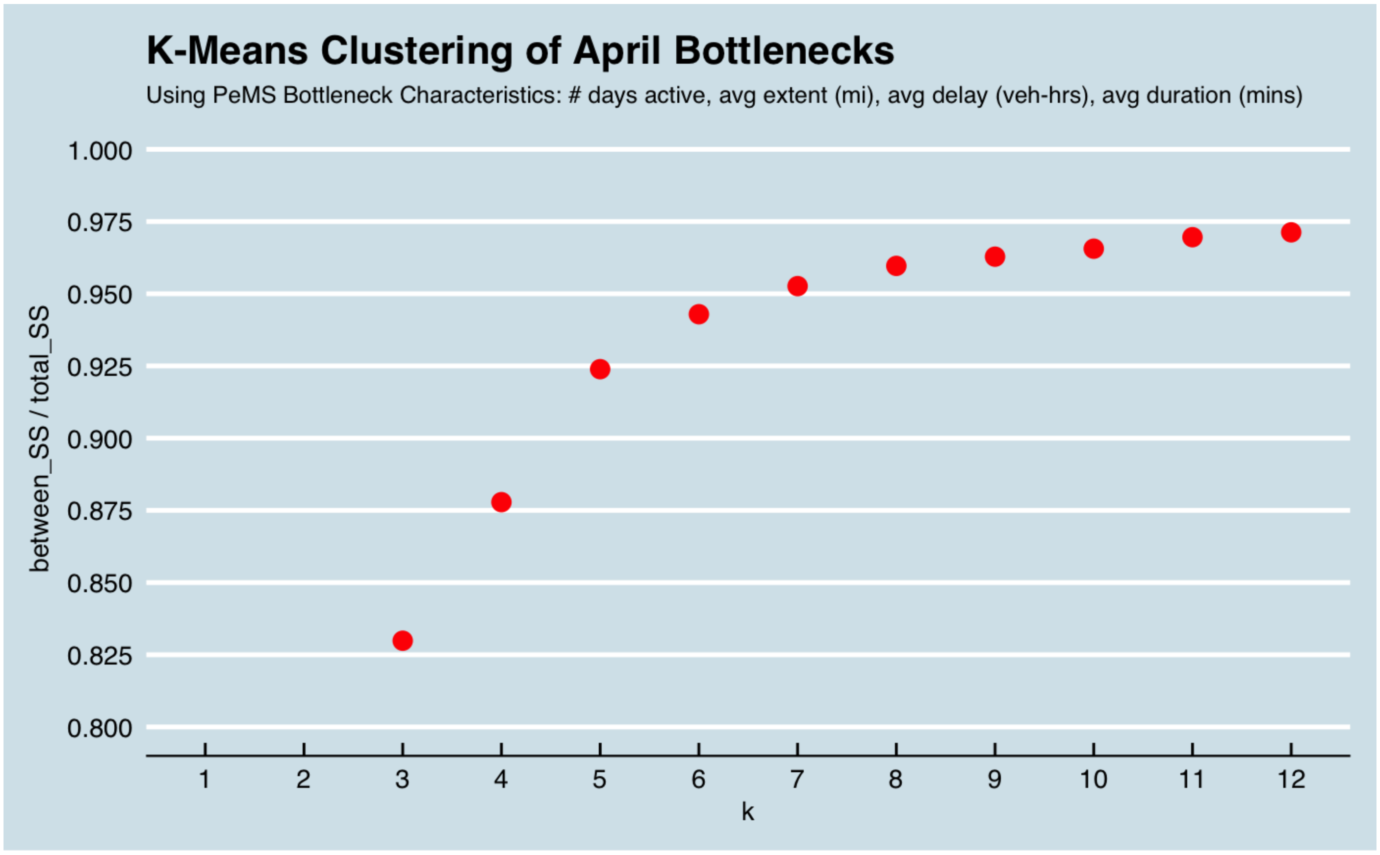
Characteristics

PeMS already reports the following four “Bottleneck Characteristics” for each detected bottleneck:

- No. Days Active
- Average Extent (Miles)
    - PeMS Description: “We measure the distance upstream that the bottleneck stretched for every 5 minutes. We then take the median of these distances for the duration of the bottleneck and call that the spatial extent of the bottleneck for that day. This column is the average of those spatial extents for all of the days that the bottleneck was active.”
- Average Delay (Vehicle-Hrs)
- Average Duration (Minutes)

While these metrics are useful, both “Average Extent” and “Average Duration” are derived from their bottleneck identification algorithm, which is based on several artificially set conditions that may be unreliable (the algorithm is discussed in the following section). The “Delay” values are also based on an artificial estimate, but it utilizes both Speed and Flow metrics from 5-minute detector data in order measure delay caused by each bottleneck. The number of days active may be useful, upon normalization, to detect whether a bottleneck is recurrent or non-recurrent.

Based on existing literature, clustering appears to be the most popular method for classifying bottlenecks among data-driven approaches (most typical approaches are highly model-driven, such as the paper by Chen et al.). The method we used is K-means clustering. The condition used to evaluate the clustering results was the percentage sum of squares explained by the algorithm (between SS / total SS). We found that (k=5) is sufficient for attaining a between-SS above 90% of the total SS, and the tested values go up to (k=12) at 97.1%.

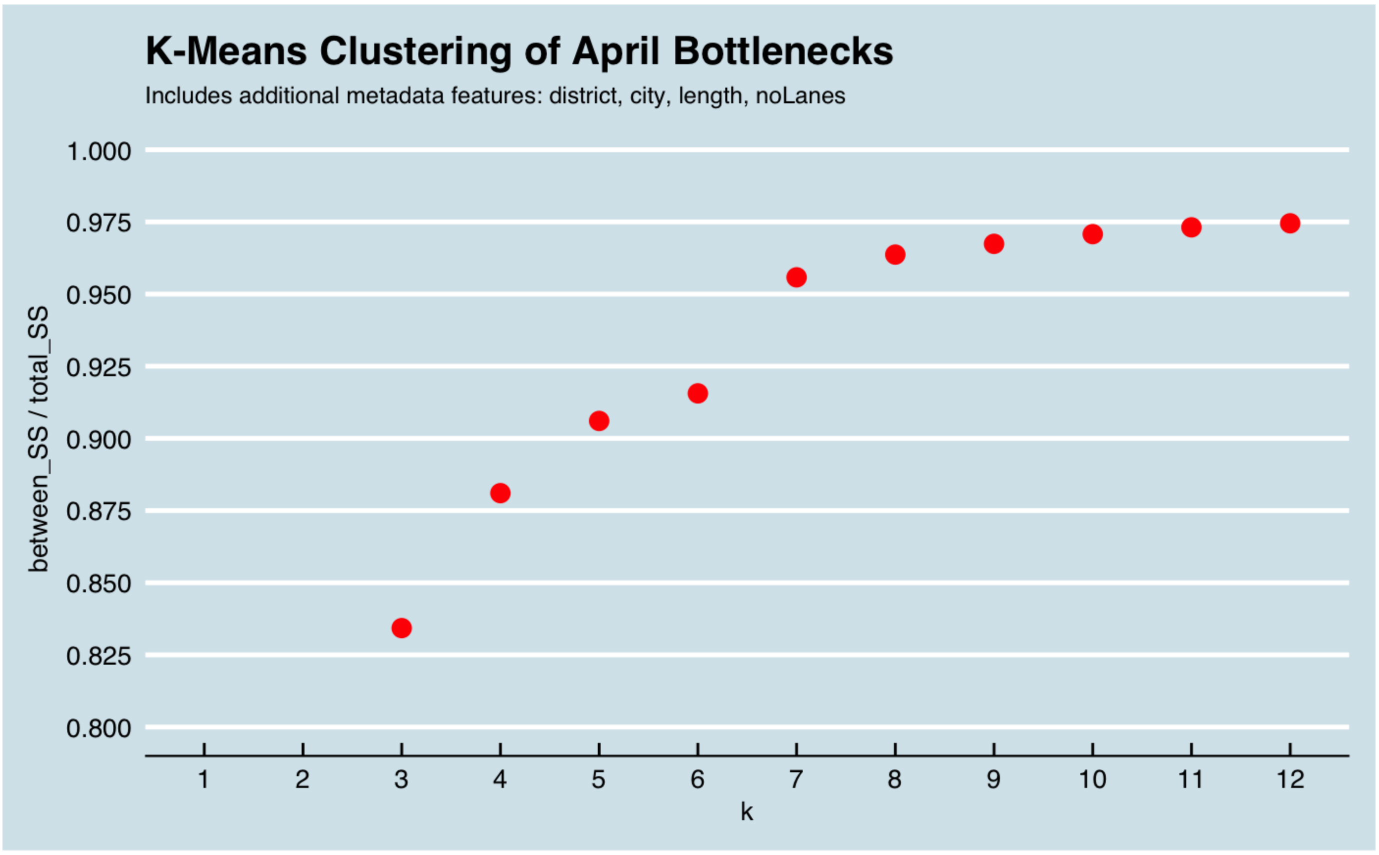


Part III: Adding Innate Station Qualities (Metadata)

It is unfortunately not possible to add features from the 5-minute detector data to the previous model because PeMS does not provide the full profile of a bottleneck; without the exact time of occurrence, upstream start point, parameters of the algorithm, etc., there is no way to assign new measurements to the detected bottlenecks. The only other source of data we have available is the Metadata table, which provides the following additional features:

```
[1] "station" "county"  "city"    "length"  "noLanes"
```

There were many bottlenecks for which the "City" field was missing, so the number of observations was reduced to (n=1649). The City and County columns are one-hot encoded before clustering.



The addition of these four metadata features only slightly altered the clustering results. Now, the 90% threshold is broken at (k=5) instead of (k=4), but by (k=12), the % SS explained converges to around 97.5%

Loading [MathJax]/jax/output/HTML-CSS/jax.js