StemGNN for Divvy Bike Re-balancing

Noah Anderson

Introduction

Divvy, Chicago's city-owned bike share program in collaboration with Lyft, has experienced significant growth since 2013. A key challenge is the imbalance in bike station capacities, with some stations frequently running empty while others overflow. To address this, Divvy employs Bike Angels, a program incentivizing users to help rebalance bikes by riding from crowded stations to those low on bikes. Despite this initiative, the dynamic nature of station statuses, especially during peak hours, highlights the need for predictive modeling.

Previous studies using regression techniques and decision trees incorporated factors like weather and past station data but achieved limited success (Krumlinde 2019). This paper explores the StemGNN model, a machine learning algorithm designed for complex network systems. Each Divvy station is treated as a network node, focusing on the percentage of occupied docks. StemGNN, proven in traffic forecasting, relies solely on past node data, providing a unique perspective for Divvy's rebalancing challenge.

This study applies StemGNN, validated in traffic prediction, to Divvy's rebalancing problem (Cao et. al 2020). Leveraging the historical data of dock occupancy, the model forecasts bike availability, potentially enhancing operational efficiency. The methods section details the adaptation of StemGNN to Divvy's data, encompassing a specific period and various neighborhoods in Chicago. The results section presents the model's performance, emphasizing how different parameters affect forecasting accuracy. This paper aims to illustrate the potential of StemGNN in urban bike-sharing systems, offering insights for future advancements in predictive modeling for urban mobility.

Methods

In this section, the methodology employed in forecasting bicycle shortages at various Divvy Stations in Chicago using a modified version of the StemGNN model in python originally developed by Microsoft (StemGNN repository, Microsoft, 2023) will be described.

The Data

The dataset used for this study was sourced from the Chicago Data Portal (City of Chicago, 2023), covering the period from March to September 2022. This dataset provided comprehensive historical information regarding the status of Divvy stations, including the availability and total number of docks at each station, recorded at 10-minute intervals.

The data queried from the Chicago Data Portal (2022) ranged from 2022-07-07 09:15:27 CDT to 2022-09-10 18:25:43 CDT. To facilitate model tuning and exploratory analysis, a focus was placed on specific neighborhoods, namely Uptown, North Center, Lincoln Square, which collectively encompassed 41 bike stations. Subsequently, the data was reformatted in R to form a T*N matrix, with 'T' representing time stamps and 'N' representing nodes. Each entry in this matrix reflected the percentage of docks in use being the predictor. The final data set had 41 nodes and 9288 rows. For more details on the data cleaning see Appendix Section B.

Table 1: Table 1: Model results by window size (Learning Rate .001)

Horizon	WindowSize	RMSE	NormalizedRMSE
3	6	5.087128	0.2137161
3	12	5.017701	0.2107154
3	18	5.049127	0.2119052
6	6	5.041065	0.2117744
6	12	5.114447	0.2147713
6	18	5.237997	0.2198242

The StemGNN model, originally proposed by Cao et al. (2020), incorporates Discrete Fourier Transform (DFT) and Graph Fourier Transform (GFT) methodologies. DFT is employed to model temporal dependencies, enabling the identification of patterns such as seasonality and autocorrelation. On the other hand, GFT is utilized to analyze interseries correlations by examining spatial interactions between nodes.

In this research, the focus was on fine-tuning two primary parameters of the StemGNN model: the learning rate and window size, while maintaining the default settings for other parameters. The learning rate influences the model's learning speed, and the window size specifies the quantity of past data points used for forecasting. Experiments were conducted with forecast horizons of 3 and 6 (equivalent to 30 and 60 minutes into the future), learning rates of 0.01, 0.001, and 0.0001, and window sizes of 6, 12, 18, were considered. All three learning rates were only run across window size 6 in the interest of time since the run time increases roughly in proportion to the increase in window size.

Root Mean Square Error (RMSE) served as the primary evaluation metric to assess model performance. It is worth noting that RMSE can be calculated in various ways for a joint time series, as it effectively consists of different models, one for each node. In this study, RMSE for the entire output, encompassing both time and space, was computed as the chosen metric for evaluation. This approach provides a comprehensive measure of model performance across both temporal and spatial dimensions. In order to assess the model performance in context with the variability of the data, RMSE normalized by the standard deviation of the test data was also included.

Results

This section presents the findings from the analysis focused on tuning the StemGNN model for forecasting bicycle dock availability at Divvy stations in Chicago. The objective was to ascertain the most effective model parameters, specifically window size and learning rate, for accurate predictions of dock availability. The impact of these parameters on the model's performance was thoroughly evaluated, leading to the identification of configurations that strike a balance between efficiency and precision. The insights gained here illuminate the complexities of model tuning and pave the way for future predictive modeling in urban mobility scenarios.

Model Tuning

The window size proved to be unimportant for model accuracy with a spread in normalized RMSE of only .002 for horizon 3 and .008 for horizon 6. As shown in Table 1, for horizon 3, a window size of 12 proved most accurate and for horizon 6, a window size of 6 performed best. The negligible improvement in performance for window size 12 for horizon 3 does not justify the fact that it doubled the run time. The key take away is that the smallest possible window size yields reasonable accuracy that either beat or closley rival models that consider more past data points.

Learning rate proved far more sensitive of a tuning parameter as evidenced by Figure 1 which plots RMSE by the exponents of learning rates on a base 10 scale. From this we can see 10^{-4} proves too slow and 10^{-2}

Table 2: Table 2: Metrics by learning rate (Horizon 3, Window Size 6

LearningRate	RMSE	NormalizedRMSE
1e-02	7.820171	0.3285344
1e-03	5.087128	0.2137161
1e-04	14.911723	0.6264587

proves too fast with a learning rate of 10^{-3} proving to yield optimal results.

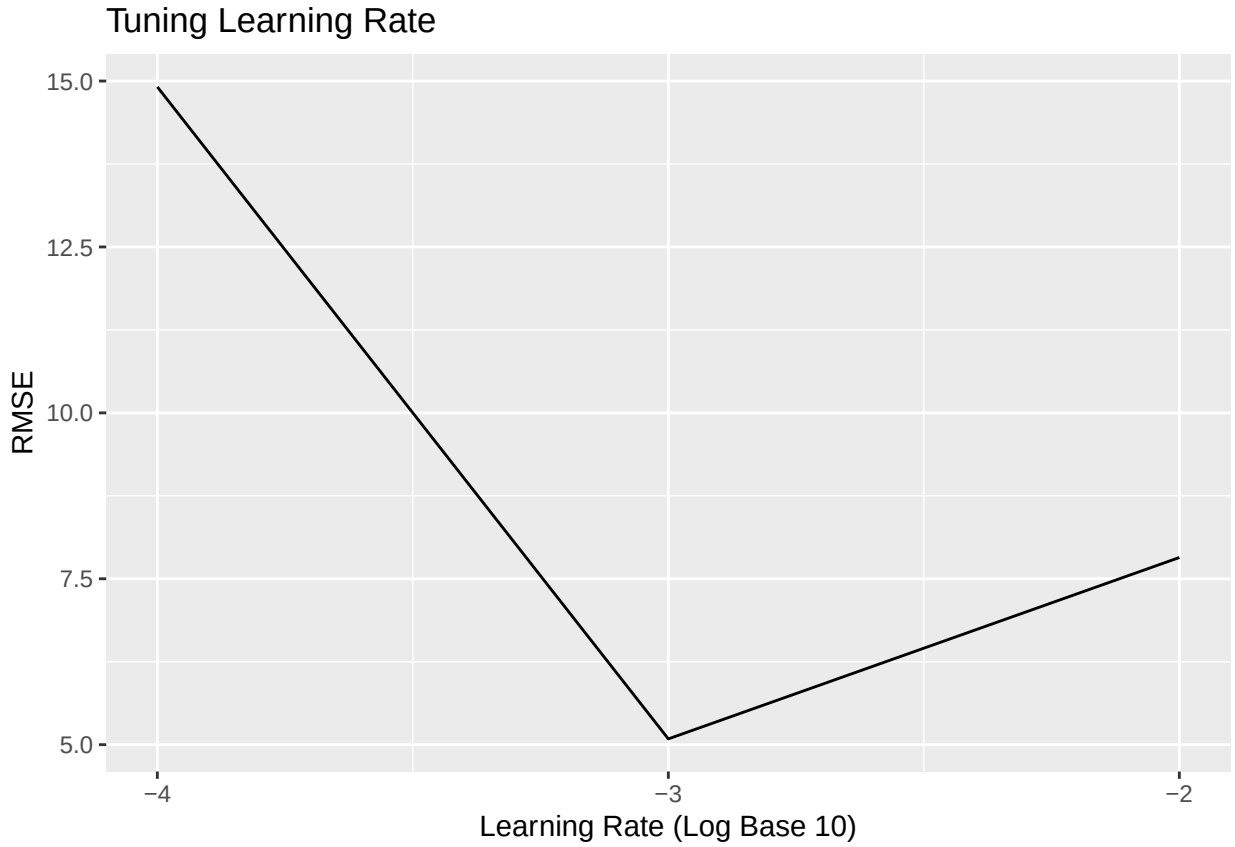

Figure 1: Figure 1: RMSE across different learning rates for the 30-minute forecast.

Most Optimal Model

The most optimal models, balancing efficiency and accuracy, used a learning rate of 10^{-3} and a window size of 6 for both 30 minute and 60 minute forecasts. This resulted in a RMSE of 5.09% and 5.04% for each model respectively. These are very optimistic results since it implies that dock availability percentage can be predicted with approximately 5% accuracy. This does not consider though the variability of the data, so RMSE was calculated as well normalizing by the standard deviation of the test data. These yield less optimistic results but still imply reasonable accuracy with results of 21.4% and 21.2% of the test datas standard deviation for horizons 3 and 6.

Discussion

The results found in this study are promising showcasing the power of StemGNN for geospatial joint time series. With the continuing growth of bike share programs more accurate rebalancing models will continue to be an important field of study cutting down on the potential cost and improving the reliability of available bikes for commuters. The results were not directly comparable to the project done by Krumlinde (2019) since different predictors were used (trips from in Krumlinde's case and percent full in this study). Another model that was in the development phase is also underway for Divvy that relies on a Poisson model. This is being developed by The Data Science For Social Good (DSSG) foundation which has proven its success with the Capital Hill Bike Share program in Washington D.C. (Henderson and Fishman 2013).

Limitiations

Both the Poisson model by DSSG and the model by Krumlinde either dealt with or plan to deal with Divvy data focusing around the Loop, the business district of Chicago. This study merely focused on four dense and populous residential neighborhoods each with their own unique restaurant and entertainment scenes. These neighborhoods were chosen due to largely not being accessible to one another by the CTA trains making Divvy bikes an optimal method of travel between neighborhoods. The idea was that these neighborhoods would be less complex than the Loop and serve as a good testing grounds for the viability of this model since they contain far less data points.

Future Steps

The next step for this model could include exploring some other tuning parameters such as Leaky Relu rate. Next adapting the code from Krumlinde and the DSSG for the percentage full data would help compare the StemGNN performance to other possible models.

The ultimate test will be to check its viability for the Loop which has the most Divvy activity and suffers from the worst balancing problems. It is the hope that this study will provide some working knowledge of optimal tuning parameters. There is no guarantee that the same parameters for the subsection examined in this project will be optimal for more complex networks, but it can at least serve as a starting point for model tuning.

Appendix

Code listed here can also be found in its full context on the authors GitHub here https://github.com/noahba65/cap-stone-PDAT. The code for the model can be found in this subdirectory located here: https://github.com/noahba65/cap-stone-PDAT/tree/main/src/stem_gnn_model.

A: Packages

```
library(RSocrata) # For reading in data from the City of Chicago
library(tidyverse) # For data cleaning and visualization
library(sf) # For geospatial data manipulation
library(stringr) # For scrubbing directories for tuning parameters
library(kableExtra) # For visualizing results
```

Packages Citations

Devlin, H., Schenk, T. Jr., Leynes, G., Lucius, N., Malc, J., Silverberg, M., & Schmeideskamp, P. (2023). RSocrata: Tools for downloading and using data from 'Socrata' data portals. R package version [1.7.15-1]. Retrieved from https://github.com/Chicago/RSocrata

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L.D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T.L., Miller, E., Bache, S.M., Müller, K., Ooms, J., Robinson, D., Seidel, D.P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., & Yutani, H. (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686. doi:10.21105/joss.01686. Retrieved from https://doi.org/10.21105/joss.01686

Pebesma, E., & Bivand, R. (2023). Spatial Data Science: With applications in R. Chapman and Hall/CRC. doi:10.1201/9780429459016. Retrieved from https://r-spatial.org/book/

Wickham, H. (2023). stringr: Simple, Consistent Wrappers for Common String Operations. R package version 1.5.1. Retrieved from https://github.com/tidyverse/stringr

Zhu, H., Travison, T., Tsai, T., Beasley, W., Xie, Y., Yu, G., Laurent, S., Shepherd, R., Sidi, Y., Salzer, B., Gui, G., Fan, Y., Murdoch, D., & Evans, B. (2021). kableExtra: Construct Complex Table with 'kable' and Pipe Syntax. R package version [1.3.4]. Retrieved from https://github.com/haozhu233/kableExtra

B: Data Setup

1: Data Acquisition

This code reads in three data sets, Boundaries - Community Areas (2018), Divvy Bicycle Stations (2022), and Divvy Bicycle Stations - Historical (2022). Boundaries - Community Areas is neccesary for geospatial data filtering in combination with Divvy Bicycle stations. Divvy Bicyle Stations, read in with R Socrata is simply a list of Divvy stations with location data which works with the Community Areas dataset. Divvy Bicycle Stations - Historical is the crucial time series data set outlined in the Methods section.

```
# Read in a local geojson file containing the shapes of Chicago Neighborhoods
chicago_ca_raw <- st_read("/Users/noahanderson/Documents/Data/Boundaries-Neighborhoods.geojson")

# Read in Divvy names and locations data set
divvy_stations_url <- "https://data.cityofchicago.org/resource/bk89-9dk7.json"
divvy_stations_raw <- read.socrata(divvy_stations_url)

# Read in historical Divvy data
divvy_raw <- read_csv("/Users/noahanderson/Documents/Data/Divvy_Bicycle_Stations_-_Historical.csv")
```

2: Data Cleaning

The Divvy Stations and Community Areas datasets are used to create `id_filter` that is used to filter the time series data only spanning the target neighborhoods.

```r
# Define vector for community areas to be queried
target_ca <- c( "Uptown", "North Center", "Lincoln Square")

# Set coordinate reference system for community areas
chicago_ca_clean <- chicago_ca_raw %>%
  st_transform(4269)

# Convert stations dataset to sf object and set coordinate reference system
divvy_stations_clean <- divvy_stations_raw %>%
  st_as_sf(coords = c("longitude", "latitude"), crs = 4269)

# Join data frames to identity stations in target community areas
divvy_stations_in_ca <- st_join(chicago_ca_clean %>% filter(pri_neigh %in% target_ca), divvy_stations_cl

# Create filter for station ID's in the target Community Areas
id_filter <- divvy_stations_in_ca$id

# Clean divvy data
divvy_10_min_clean_percent <- divvy_raw %>%
  rename( percent_full = `Percent Full` ) %>%
  filter( ID %in% id_filter) %>%
  select(Timestamp, ID, percent_full) %>%
  mutate(Timestamp = as.POSIXct(Timestamp, format = "%m/%d/%Y %I:%M:%S %p"))


# Pivot data to wide T*N format
divvy_10_min_wide_percent <- divvy_10_min_clean_percent %>%
  pivot_wider(names_from = ID, values_from = percent_full) %>%
  arrange(Timestamp) %>%
  select(-Timestamp)

# Remove column names
colnames(divvy_10_min_wide_percent) <- NULL

# Save csv to StemGNN sub directory
save_csv(divvy_10_min_wide_percent, "/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/stem_gnn_m
```

C: Evaluation

1: Custom Functions

This code builds custom functions for reading in model output data specific to the structure of the project repository along with functions for computing model metrics.

```r
read_error_and_target_files <- function(base_dir) {
  # List all subdirectories within the base directory
  subdirs <- list.dirs(base_dir, full.names = TRUE, recursive = FALSE)

  # Initialize a list to store the data frames
  data_list <- list()

  # Loop through each subdirectory
```

```
  for (dir in subdirs) {
    # Construct file paths
    error_file_path <- file.path(dir, "predict_abs_error.csv")
    target_file_path <- file.path(dir, "target.csv")

    # Check if the files exist and read them
    if (file.exists(error_file_path) && file.exists(target_file_path)) {
      error_df <- read_csv(error_file_path, col_names = FALSE)
      target_df <- read_csv(target_file_path, col_names = FALSE)

      # Add the data frames to the list
      data_list[[basename(dir)]] <- list(Error = error_df, Target = target_df)
    }
  }

  return(data_list)
}

# Function to calculate RMSE and normalized RMSE by standard deviation
calculate_rmse <- function(data_list, dir_name) {
  error_df <- data_list$Error
  target_df <- data_list$Target

  # Calculate RMSE
  rmse <- sqrt(mean(as.matrix(error_df^2)))

  # Calculate the standard deviation of the target values
  std_dev <- sd(as.vector(as.matrix(target_df)))

  # Normalize RMSE by the standard deviation
  normalized_rmse <- rmse / std_dev

  return(data.frame(Directory = dir_name, RMSE = rmse, NormalizedRMSE = normalized_rmse))
}

# Function to extract window size and horizon from directory name
extract_info <- function(dir_name) {
  window_size <- as.numeric(str_extract(dir_name, "(?<=window_size_)[0-9]+"))
  horizon <- as.numeric(str_extract(dir_name, "(?<=horizon_)[0-9]+"))
  learning_rate <- as.numeric(str_extract(dir_name, "(?<=lr_)[0-9\\.e-]+"))
  return(c(WindowSize = window_size, Horizon = horizon, LearningRate = learning_rate))
}
```

2: Evaluating Metrics

This code reads in and computes model metrics that are presented in the study.

```
base_dir <- "/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/stem_gnn_model/output/divvy_10_min

# Reads in absolute error datasets
abs_error_and_target_data <- read_error_and_target_files(base_dir)

# Apply the RMSE calculation to each dataframe and combine the results into a dataframe
```

```r
rmse_results_df <- map_df(names(abs_error_and_target_data), ~calculate_rmse(abs_error_and_target_data[[


# Add columns for window size and learning rate
rmse_results_df_clean <- rmse_results_df %>%
  mutate(WindowSize = map_dbl(Directory, ~extract_info(.x)["WindowSize"]),
         Horizon = map_dbl(Directory, ~extract_info(.x)["Horizon"]),
         LearningRate = map_dbl(Directory, ~extract_info(.x)["LearningRate"])) %>%
  select(-Directory)

# Select and arrange data for final format
metrics_df <- rmse_results_df_clean %>%
  select(Horizon, LearningRate, WindowSize, RMSE, NormalizedRMSE) %>%
  arrange(Horizon, WindowSize)

# Save results
write_csv(metrics_df, "/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/eval/data/percent_metrics
```

D Visualizations

This code reads in metrics data created in the previous section and builds the vizualistions used in this paper.

```r
# Read in metrics data
percent_metrics_df <- read_csv("/Users/noahanderson/Documents/GitHub/cap-stone-PDAT/src/eval/data/percen

# Create table 1
kable( percent_metrics_df %>% filter(LearningRate == .001) %>% select(-LearningRate), caption = "Table

# Create table 2
kable( percent_metrics_df %>% filter(WindowSize == 6, Horizon == 3) %>% select(-Horizon, -WindowSize), c

# Create figure 1
percent_metrics_df %>%
  filter(Horizon == 3, WindowSize == 6) %>%
  ggplot() +
  geom_line(aes(x = log10(LearningRate), y = RMSE)) +
  scale_x_continuous(breaks = c(-4, -3, -2)) +
  ggtitle("Tuning Learning Rate") +
  xlab("Learning Rate (Log Base 10)") +
  labs(caption = "Figure 1: RMSE across different learning rates for the 30-minute forecast.")
```

References

Cao, D., Wang, Y., Duan, J., Zhang, C., Zhu, X., Huang, C., Tong, Y., Xu, B., Bai, J., Tong, J., & Zhang, Q. (2020). Spectral temporal graph neural network for multivariate time-series forecasting. Advances in neural information processing systems, 33, 17766-17778. https://doi.org/10.48550/arXiv.2103.07719

City of Chicago. (2018, December 18). Boundaries - Community Areas (current). Retrieved [2023

October 30], from https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-Community-Areas-current-/cauq-8yn6

City of Chicago. (2022, September 10). Divvy Bicycle Stations. Retrieved [2022 September 10], from https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations/bbyy-e7gq

City of Chicago. (2022, September 10). Divvy Bicycle Stations - Historical. Retrieved [2023 November 26], from https://data.cityofchicago.org/Transportation/Divvy-Bicycle-Stations-Historical/eq45-8inv

City of Chicago. "Divvy Trips." Data published by City of Chicago. Accessed on November 27, 2023. Available at: https://data.cityofchicago.org/Transportation/Divvy-Trips/fg6s-gzvg.

Henderson, J., & Fishman, A. (2013, August 9). Divvy: Helping Chicago's New Bike Share Find Its Balance. Data Science For Social Good. https://www.dssgfellowship.org/2013/08/09/divvy-helping-chicagos-new-bike-share-find-its-balance/

Krumlinde, Z. (2019, July 22). Using Machine Learning to Predict Hourly Divvy Bike-Sharing Checkouts per Station. Towards Data Science. https://towardsdatascience.com/predicting-hourly-divvy-bike-sharing-checkouts-per-station-65b1d217d8a4

Microsoft. (2023). StemGNN: Spectral Temporal Graph Neural Network. GitHub. Retrieved from https://github.com/microsoft/StemGNN