
I) Algorithm Steps: [python3 Predict.py]

1) Converts the whole 40km x 40km area into a matrix(r,c).
2) From the provided data frame I obtain the avg_time taken to travel within a specific cell(latitude-longitude coordinate) in the matrix.
3) I have built 4 such time matrices because bus speed varies based on the time of day.
4) Traverses the matrix from the input to the output cell and performs a summation of time taken in all 4 matrices.
5) Update these weights(matrices) based on the loss.
6) Store these matrices and use them for prediction.

II) Algorithm Details:

1) Convert the whole area to a matrix of rows and columns, the dimensions are determined by the user. I have used rows = 32, columns = 32.

- convert_coordinates(coords1,coords2) - Given a latitude and longitude coordinates it obtains the distance between them.
- get_distance(lat_,long_,height,width) - Returns the diagonal distance between a specific latitude and longitude.
- calculate_dimension(r,c) - Returns the size of a single cell of the matrix.

2) Obtains the average time it takes for all buses in a specific cells at specific timings [morning,afternoon,night,otherwise]

- find_location(x_start,x_end,y_start,y_end) - This function will obtain all the bus speeds within a given latitude and longitude. It will then calculate and return the average speed within these coordinates.
- build_location_matrix(r,c) - Finally returns the 4 time matrices by calling the previous functions.

3) Training the data

- train_data(r,c) - Function to pass the input and output dataframe to perform training.

- get_index(x,y) - Given a specific latitude and longitude coordinate we identify which part of the grid it belongs. Obtain the start and end position in the grid.
- box_cox_path_of_points(X,Y,train) - Traverse from one cell to another in the time matrix and then perform a summation of the values present in the individual cells.
- time_taken(P1,P2,P3,P4) - Returns the time taken to travel from one coordinate to another in all 4 time matrices.
- calculate_path(input_data,output,train) - Compares the predicted time with the actual time taken.
- update_weight_matrix(positions,lr) - Gradients are calculated based on the loss value and respective cell values in all 4 matrices are updated.
- store_matrix_check() - Save the learnt weights at the end of training.

III) Other functions :

1) modelflow() - Main function.
2) estimated_time_travelled(df, dfInput) - Function where only the test data is passed and the predicted values are obtained and stored in a dataframe.
3) relu(x) - Returns max(0,x)
4) invboxcox(y,ld) - Performs inverse of the box_cox transformation.
a) Data Analysis Functions -
1) data_summary() - Print the head of the different data frames
2) get_error_prone_paths(all_error_prone_paths) - This function was used to analyze which locations in the location grid give the most inaccurate results.

The data_transformation.py function is used to perform data transformation of the matrices. The performance does not improve significantly after normalizing the weights so it is not required for training but can be integrated.
