NEXUS CHALLENGE 2021
ADDRESSING NETWORK IMBALANCES REPORT

Qi Wen, Jason Tran, Daniel Mai
Department of Computer Science
University of Manitoba
Winnipeg, Canada
wenq@myumanitoba.ca, tranndt@myumanitoba.ca, maith@myumanitoba.ca

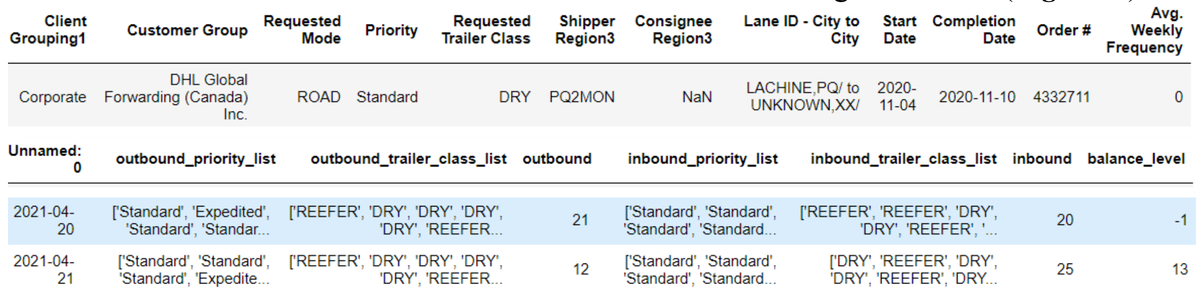
I. Introduction

The challenge offered by Bison Transport Company is about how we balance the inbound and outbound of current days and predict what would be the inbound and outbound freights for the future. Our goal of this paper is to predict the future traffic trend of a certain region, e.g. Quebec, in terms of inbound freight and outbound freight, and apply our optimization algorithm, a novel algorithm called k-local greedy algorithm to minimize the daily/weekly imbalance based on the prediction output. In particular, we would analyze the pattern of the given dataset and optimize the imbalance level by moving orders within 24 hours. Based on those criteria, our model would give the prediction of what would be the inbound/outbound freight for each day in Quebec in the future.

II. Methodology

1. Analyze and preprocess the dataset

Our prediction is based on Long short-term memory (LSTM) , which is an artificial recurrent neural network (RNN) architecture used in the field of machine learning, thus, a smaller dataset will save us a lot of computation time. We eliminated columns that do not provide us much information, e.g., 'Requested Mode' only contains the 'Road' type. We also cut off the records not related to the 'PQ' region. Furthermore, we also find empty cells in the dataset and observe them to decide whether we will drop it or modify it with proper information. Importantly, we replace the column 'Shipper Region3' and 'Consignee Region3' by the 'Inbound' and 'Outbound' columns respectively to indicate the number of orders going out and to Quebec. Finally, each row in the preprocessed dataset contains one day's traffic information. With this approach, we successfully reduced the size of the dataset to about 50KB from 1300+KB original dataset.(***Figure 1***)



Client Grouping1	Customer Group	Requested Mode	Priority	Requested Trailer Class	Shipper Region3	Consignee Region3	Lane ID - City to City	Start Date	Completion Date	Order #	Avg. Weekly Frequency
Corporate	DHL Global Forwarding (Canada) Inc.	ROAD	Standard	DRY	PQ2MON	NaN	LACHINE,PQ/ to UNKNOWN,XX/	2020-11-04	2020-11-10	4332711	0

Unnamed: 0	outbound_priority_list	outbound_trailer_class_list	outbound	inbound_priority_list	inbound_trailer_class_list	inbound	balance_level
2021-04-20	['Standard', 'Expedited', 'Standard', 'Standar...	['REEFER', 'DRY', 'DRY', 'DRY', 'DRY', 'REEFER...	21	['Standard', 'Standard', 'Standard', 'Standard...	['REEFER', 'REEFER', 'DRY', 'DRY', 'REEFER', '...	20	-1
2021-04-21	['Standard', 'Standard', 'Standard', 'Expedite...	['REEFER', 'DRY', 'DRY', 'DRY', 'DRY', 'REEFER...	12	['Standard', 'Standard', 'Standard', 'Standard...	['DRY', 'REEFER', 'DRY', 'DRY', 'REEFER', 'DRY...	25	13

Figure 1. Shows the original dataset(first) and preprocessed dataset(second)

2. Data model

We apply the LSTM model for the time series forecasting, i.e., predict the future trend of the traffic. In particular, we predict the cumulative imbalance level of the future N numbers of days. Parameters used in the model:

- Observation of the sample: 60
- Sequential() method to build model
- Dropout 0.2 regularisation
- Epochs: 50 - 100

Generally speaking, our prediction accuracy is around 90%(explained in Section 4) depending on how the users define the fluctuation range(N). After obtaining the predicted trend, we apply a k-local greedy algorithm to balance the inbound and outbound for each day.

3. K-local greedy algorithm

We propose a method to improve the balance level of the current schedule. The idea being, "Given a number k, the algorithm will go through every group of k consecutive days in the schedule and try to even out the imbalances of each and every day", hence the name k-local greedy algorithm.

In detail, as we go through each day in the original schedule, we will look ahead at the next k-days forecast and then re-tune the inbound and outbound schedule so that we achieve the k-days average balance level for our current day and then iteratively to all the other days in k-days group. To achieve the desired balance level for any particular day, say May 1st, we will be looking to move certain inbound or outbound orders between itself and the very next day (within 24 hours) May 2nd, or a combination of both. Suppose we have been able to move such a combination of orders, once May 1st is balanced, we pop it out of the list and move on to resolve the imbalances of the next day, May 2nd - partly caused by the exchanges with May 1st - and try to get its balance level to the k-day average level by moving orders from and to the day after that, May 3rd. Once we finish balancing every consecutive pair of days in the k-day local group, we move on to the next k-days group that starts on May 2nd, and repeatedly resolve the imbalances.

One important step is to determine how many orders of each type we want to move to achieve the desired balance level. For example, taking in 3 inbound orders, or taking in 1 inbound order and deferring 2 outbound orders to the next day will all achieve the same effect of increasing the balance level by 3. We found that traffic volume is a reasonable indicator as to which combination we should choose. Particularly, if current day's traffic is heavier in both directions (inbound and outbound) than the next's, we will prioritize the option of deferring inbound/outbound traffic to achieve the desired effect; in contrast, if current day traffic is lighter than the next's, we will prioritize the option of taking in inbound/outbound orders to achieve the desired effect. If we get a mixed situation, traffic is heavier in one direction but lighter in the other, we will try to do both: first take in some inbound/outbound orders then try to defer some outbound/inbound orders. This way we will also be able to achieve a balance of traffic volume across days, on top of the desired balance level between inbound and outbound traffic for every day.

After we figure the number of orders in each direction we need to move in between the pair of days, another important step is to determine which of the orders on that day should be moved. Not all orders can be moved to the next/previous day, because we risk setting unrealistic duration (too long/short). To preserve the schedule integrity, we have a mechanism in place to ensure that we only pick orders whose duration in the new schedule will be in the expected range of the original schedule. We also further optimize by ordering the movable orders list so that the most flexible orders are taken first when we are looking to shorten the duration, and the most constrained orders are taken first when we are looking to extend the duration. This likely increases the movability of our orders, and the more movability there is in our schedule, the better the overall performance.

```python
# Pseudo-code
# Our main k-local greedy algorithm that re-tunes our schedule
def k_local_greedy(schedule,k):
    for day in schedule.all_days:
        k = min(schedule.num_days-day,k) # Pick the number of days left if < k
        local_k_days = schedule[day:day+k]
        bal_val = average_imb(local_k_day)
        while local_k_days.size > 1:
            curr,next,last = local_k_days[0], local_k_days[1], local_k_days[-1]
            schedule.balance(curr,next,bal_val)
            local_k_days.pop(curr)
    return schedule
```

4. Evaluation and Visualization

a. Prediction accuracy

We separate the dataset into two parts - training dataset and testing dataset with 70% and 30% of total size of dataset respectively. In particular, 538 days of record will be trained to predict 230 days of future cumulative imbalance level(CIL). ***Figure 2*** illustrates the overall prediction result.

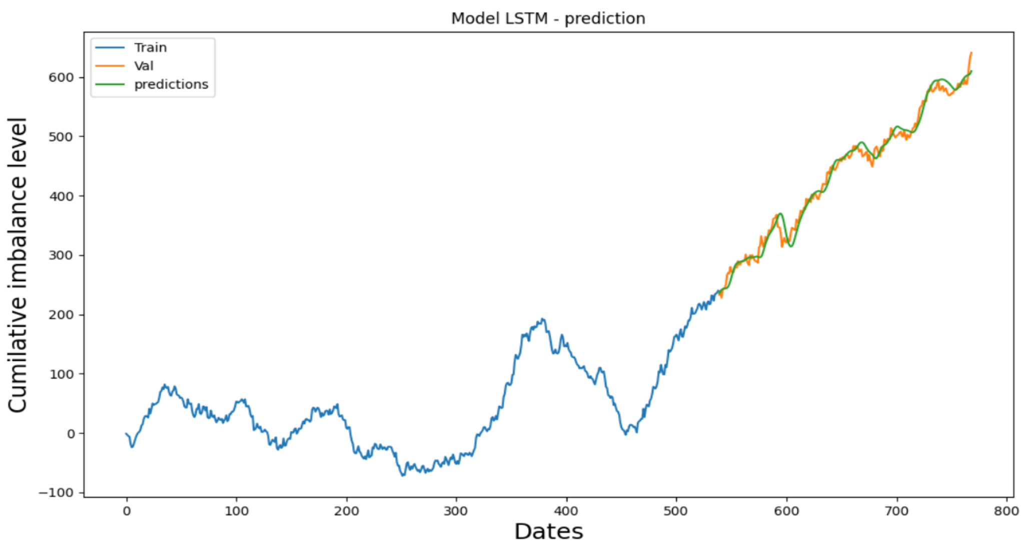
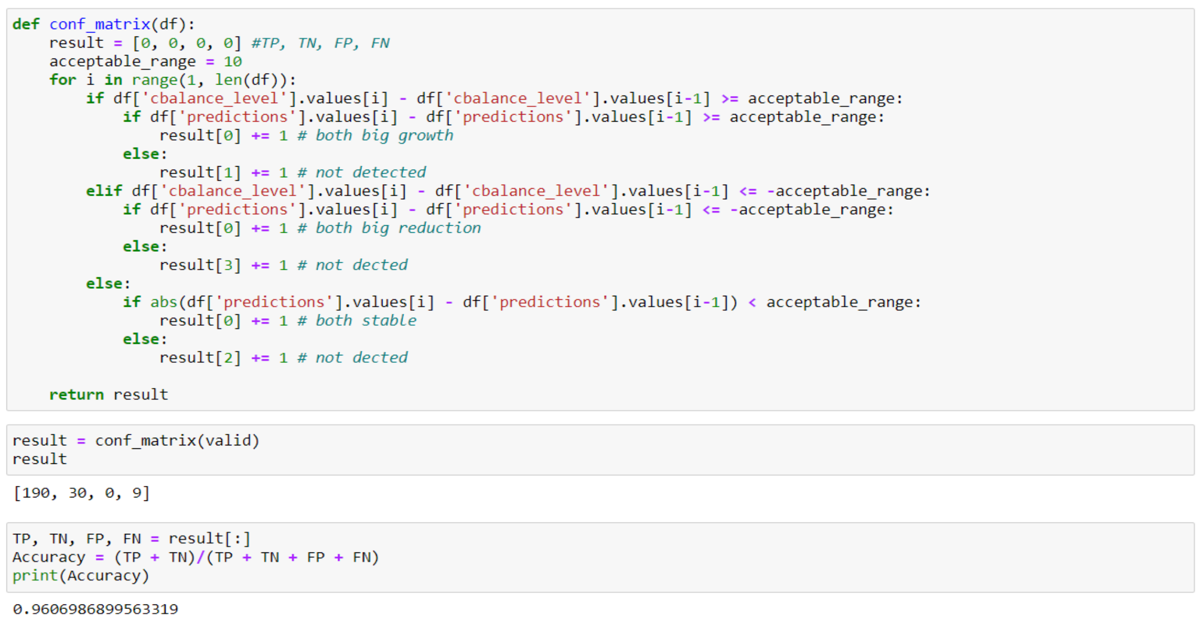


Figure 2. training dataset, original dataset, our prediction



```
def conf_matrix(df):
    result = [0, 0, 0, 0] #TP, TN, FP, FN
    acceptable_range = 10
    for i in range(1, len(df)):
        if df['cbalance_level'].values[i] - df['cbalance_level'].values[i-1] >= acceptable_range:
            if df['predictions'].values[i] - df['predictions'].values[i-1] >= acceptable_range:
                result[0] += 1 # both big growth
            else:
                result[1] += 1 # not detected
        elif df['cbalance_level'].values[i] - df['cbalance_level'].values[i-1] <= -acceptable_range:
            if df['predictions'].values[i] - df['predictions'].values[i-1] <= -acceptable_range:
                result[0] += 1 # both big reduction
            else:
                result[3] += 1 # not dected
        else:
            if abs(df['predictions'].values[i] - df['predictions'].values[i-1]) < acceptable_range:
                result[0] += 1 # both stable
            else:
                result[2] += 1 # not dected

    return result
```

```
result = conf_matrix(valid)
result
```

```
[190, 30, 0, 9]
```

```
TP, TN, FP, FN = result[:]
Accuracy = (TP + TN)/(TP + TN + FP + FN)
print(Accuracy)
```

```
0.9606986899563319
```

Figure 3. Method implementing confusion matrix and its results

Based on our prediction model, we expect to notice if there will be a large CIL increase or decrease, that is if we can dect ± N range of fluctuation in traffic on the next day. For example, if the real CIL decreases by 10 tomorrow, does our predicted data for the corresponding date also decrease by 10 or more. If so, we can take action today, e.g., we can back up some drivers because there will be more incoming traffic tomorrow. So we define TP(True Positive), TN(True Negative), FP(False Positive), FN(False Negative) as follows (***Table 1***): TP - best cases, TN - acceptable cases, FP - non-acceptable cases, FN - worst cases. The accuracy can be defined as (TP + TN)/(TP + TN + FP + FN) and we apply N=10 that is fluctuation range=10, as a result, 210 out of 229 predictions(the size of testing data 230-1 as we are starting from the second index to do the comparison) are in either TP or TN that is more than 96% of accuracy(***Figure 3***).

	real CIL grows more than N	real CIL decreases more than N	real CIL fluctuates in ±N
predicted CIL grows more than N	TP	FN	FP
predicted CIL decrease more than N	FN	TP	FN
predicted CIL fluctuates in ±N	FP	FN	TN

Table 1. Classification of TP, TN, FP, FN

b. k-local greedy algorithm

From ***Figure 4,*** we can see our algorithm improve imbalances level across the dataset, and the overall distribution smoothens and approaches the zero balance level.

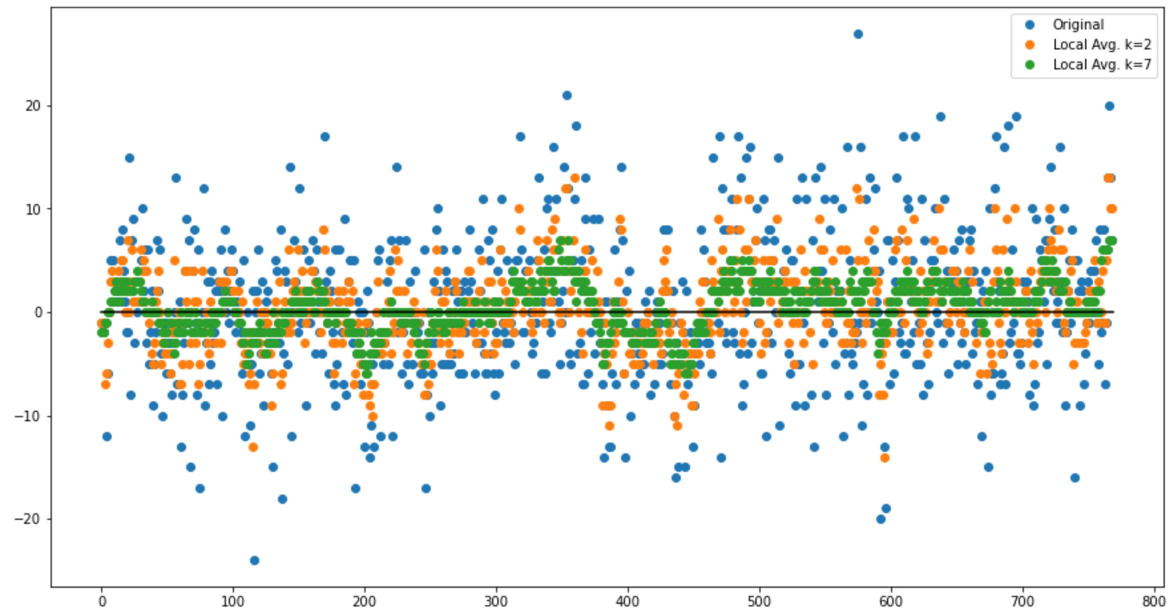


Figure 4. The daily imbalance level - the blue points are the original imbalance level, orange points and green points are the imbalance level after applying the algorithm with k=2 and k=7

The dataset's standard deviation can tell us how far away the imbalance levels are from the mean, and since the mean imbalance of the dataset is close to 0, we can conveniently use this as a success metric to help us understand how our algorithm helps improve the overall imbalance level. In the case where k=7, the algorithm produces a schedule with a standard deviation in the imbalance level of 2.4310, a 66% reduction compared to the initial value of

```
Standard Deviation
Org: 7.203951411331257
k=2: 4.151632473796118
k=3: 3.320255363031487
k=4: 2.984994607852571З
k=5: 2.716405282546249
k=6: 2.5563578076937437
k=7: 2.4310420072382057
```

7.2039.

From ***Figure*** *5*, we can also see that compared to the original dataset, not only are the inbound- outbound traffic count pairs closer to the (diagonal) balance line, indicating better balance, but they are also form a more cohesive cluster, indicating more harmonious day-to-day traffic volumes, and less extreme variations.

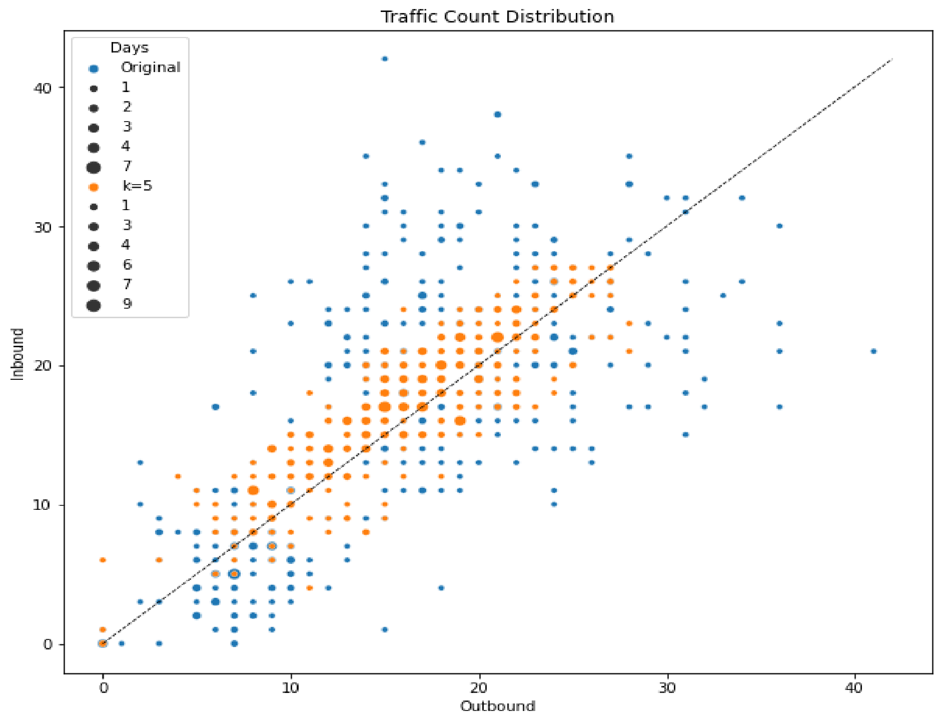


Figure *5*. Inbound - Outbound traffic pair in the original dataset and after we applied the algorithm where k=5

III. Conclusion

In this paper, we presented a data science solution for addressing network imbalances. Specifically, our approach is divided into two parts: prediction and optimization. In the prediction part, we applied the LSTM network to predict the future trend of the traffic and achieved approximately 96% of accuracy when N=10. In other words, if the future traffic increases or decreases significantly, our model can successfully predict the trend. Furthermore, even if we set N=5, it still provides over 80% of accuracy. As for the optimization part, we applied the k-local greedy algorithm that uses a greedy strategy of optimizing each day's imbalances based on the immediate future forecast, and achieved an improvement of 66% in regards to the dataset standard deviation when k = 7. We also find that greater k values means smaller standard deviations in the resulting schedule, but there is a point of diminishing return, so time-efficiency tradeoffs should be taken into consideration. As a future work, we expect that a combination of both these approaches can lead to interesting future findings in coming up with a better predictive algorithm that can also provide a good strategy to better manage the imbalance.