Project Report

On

Travel Time Optimization using Machine Learning

Submitted by

Vikash Kumar 170001054

Vikram Kushwaha 170001055

Computer Science and Engineering

3rd year

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Spring 2019

Introduction:

Objective:

The goal of our project is to optimize travel routes for a delivery vehicle by using machine learning model predictions. This is a two-component problem: first, We train a machine learning model on the data to predict how long it will take a delivery vehicle to go from point one point to another, and I feed these predictions into a genetic algorithm which decides which is the most time efficient visit order for a given set of points.

Motivation:

Why are we using Machine Learning for optimization problem? We may have a minimization or a maximization problem to solve and that we can solve using optimization techniques, but what if we don't have the proper data and parameters (let's say the travel time between two geospatial locations in our case), then Machine Learning comes into play. We can use ML to predict data and parameters which we don't know beforehand.

Algorithm Analysis:

Problem Definition:

For given N geospatial locations, time to travel to each location once is given by the function f(). Our problem is to minimize the value of f(). Mathematically, we can denote it as follows:

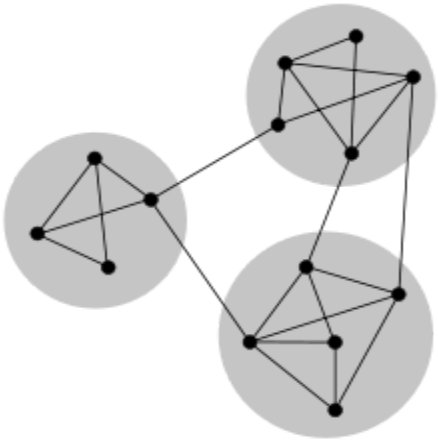
$$\min f = \sum t_i \quad \text{where } i \in \{1,2,3\ldots\ldots, N-1\}$$

$$\text{subject to: each } t_i > 0;$$

where t_i is the time taken to travel between point a point A to point B.

Consider a scenario: a UPS driver with 25 packages has 15 trillion possible routes to choose from. And if each driver drives just one more mile each day than necessary, the company would be losing $30 million a year.
While UPS would have all the data for their trucks and routes, there is no way they can run 15 trillion computations per each driver with 25 packages. However, this traveling salesman problem can be approached with something called the “genetic algorithm.” The issue here is that this algorithm requires having some inputs, say the travel time between each pair of locations, and UPS wouldn’t have this information, since there are more than even trillions of combinations of addresses. But what if we use the predictive power of machine learning to power up the genetic algorithm?

Available Algorithm : Kruskal’s Algorithm



Consider an undirected graph G(v,e), where v contains the set of N geospatial locations and ‘e’ is the set containing travel time between each pair of points. Time complexity of Kruskal’s is given by O(e*log v) where “e” can be at most “v^2 ”.

One of the limitations of Kruskal’s Algorithm is that it has only one edge between a pair of points. But in real world scenarios, there are multiple paths between two locations. Iterating through those path to find the shortest will further increase the time complexity of the problem.Since in paths between a pair of points can be infinite, Kruskal becomes an inefficient algorithm for our problem.

The other limitation of Kruskal's Algorithm is that the result obtained from does not contain a cycle. Whereas in our case, we can have a cycle to obtain the minimum path.

```
KRUSKAL(G):                                          (Wikipedia)
      A = ∅
      foreach v ∈ G.V:
            MAKE-SET(v)
      foreach (u, v) in G.E ordered by weight(u, v), increasing:
          if FIND-SET(u) ≠ FIND-SET(v):
                  A = A ∪ {(u, v)}
                  UNION(FIND-SET(u), FIND-SET(v))
      return A
```

Algorithm Design: Genetics Algorithm

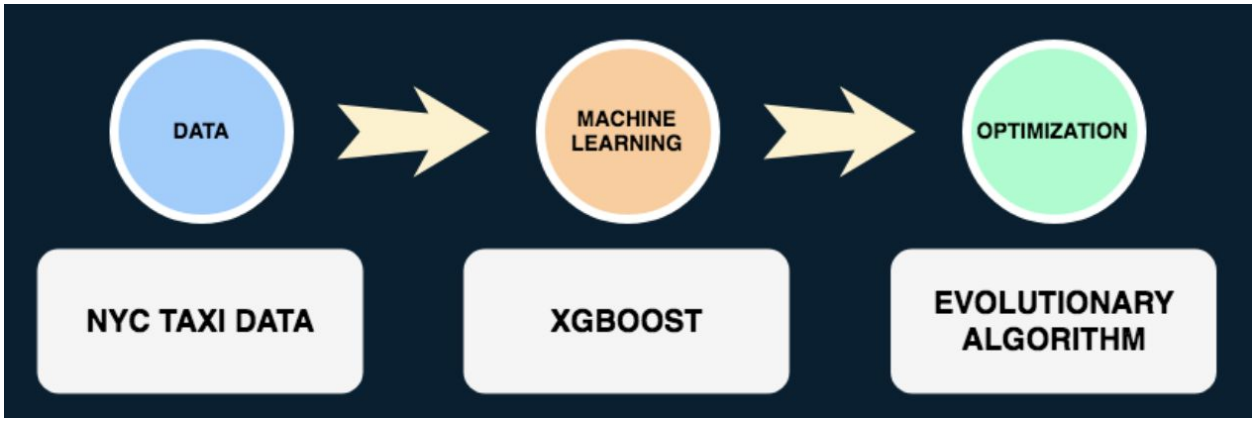
In the genetic algorithm, a population of candidate solutions to an optimization problem evolves toward better solutions, and each candidate solution has a set of properties which can be mutated and altered. Basically, we start with a random solution to the problem and try to “evolve” the solution based on some fitness metric. The result is not guaranteed to be the best solution possible, but it should be close enough.

Let’s say we have 11 points on the map. We would like our delivery truck to visit all these locations on the same day, and we want to know the best route. However, we don’t know how long it will take the driver to go between each point because we don’t have the data for all address combinations.This is where the machine learning part comes in. With our predictive model, we can find out how long it will take for a truck to get from one point to another, and we can make predictions for each pair of points.

Machine learning Model: XGBoost

```
gbm = xgb.train(params,
                dtrain,
                num_boost_round = nrounds,
                evals = watchlist,
                verbose_eval = True)
```

When we use our model as a part of the genetic algorithm, what we start with is a random visit order for each point. Then, based on the fitness score being the shortest total time traveled, the algorithm attempts to find a better visit order, getting predictions from the machine learning model. This process repeats until we figure out a close to ideal solution.




```
def fitness_score(guess):  //Gives fitness score to each population(paths)
            """ Loops through the points in the guesses order and calculates
                how much distance the path would take to complete a loop.
                Lower is better. """
                score = 0
                for ix, point_id in enumerate(guess[:-1]):
                    score += travel_time_between_points(coordinates[point_id],
                coordinates[guess[ix+1]],                    11, my_date)
return score
```

The genetic algorithm loops through all the populations to find the populations with good fitness score to pass into the next generation. The above process is repeated multiple times to find the best population.

Implementations and Results:

Program Used:

A. Anaconda Navigator
B. Jupyter Notebook
C. Excel

Downloads and Installation Process:

1. Download and install Anaconda. Inside Anaconda Navigator, install Jupyter Notebook.
2. Code for both XGBoost and Genetic Algorithm was written in the Jupyter Notebook. First we had to install the following packages inside Anaconda Environment:
    - Conda : Os-agnostic system level binary package an environment manager
    - Geopy : GeoPy is a Python library that makes geographical calculations easier for the users
    - Openpyxl : python library to read write excel file
    - py-xgboost : XGBoost is an optimized distributed gradient boosting library to implement machine learning algorithms under the Gradient Boosting framework.
3. Libraries imported inside XGBoost Code are:
    - pandas : pandas is a software library for data manipulation and analysis
    - pickle : Python pickle module is used for serializing and de-serializing a Python object structure.
    - xgboost : XGBoost is an open-source software library which provides a gradient boosting framework
    - numpy : Numpy is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc.
    - geopy : geopy makes it easy for Python developers to locate the coordinates of addresses, cities, countries, and landmarks across the globe using third-party geocoders and other data sources. From geopy, geocoder class was used for our purpose
    - sklearn : Scikit-learn is a free software machine learning library for the Python programming language.
4. Libraries Imported for Genetic Algorithm Code are :
    - Pandas, Pickle, XGBoost, Numpy, Geopy
    - Copy : Used to make ‘clone’ and ‘real copies’ of objects
    - datetime : datetime module supplies classes for manipulating dates and times in both simple and complex ways

Execution and Explanation of code:

Both the XGBoost_Model.ipynb and the genetic_algorithm.ipynb code is properly commented for ease of understanding, please refer to the actual code here at GitHub to get the detailed insight about our code.

Here we are providing an overview of our code in a brief manner.

XGBoost jupyter notebook(file named Train XGBoost_Model.ipynb) is executed on train.csv dataset file containing 14,58,644 training examples. An output named xgb_model.sav is created which is further imported in Genetic Algorithm notebook for implementation.

Since we have implemented in jupyter notebook, we can execute the individual blocks of the code.

The XGBoost_Model.ipynb code was executed in the following order.

- First all the libraries were imported as mentioned above.

```
import pandas as pd
import numpy as np
import xgboost as xgb

import pickle
from geopy.geocoders import Nominatim
from sklearn.model_selection import train_test_split

pd.set_option('display.max_columns', None)
```

- Then the data was imported from train.csv and we took a sample look at it to make sure that our data is read properly by the model.

  Our data fields in train.csv are organized as follows

  - id - a unique identifier for each trip
  - vendor_id - a code indicating the provider associated with the trip record
  - pickup_datetime - date and time when the meter was engaged
  - dropoff_datetime - date and time when the meter was disengaged
  - passenger_count - the number of passengers in the vehicle (driver entered value)
  - pickup_longitude - the longitude where the meter was engaged
  - pickup_latitude - the latitude where the meter was engaged
  - dropoff_longitude - the longitude where the meter was disengaged

- dropoff_latitude - the latitude where the meter was disengaged
- store_and_fwd_flag - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- trip_duration - duration of the trip in seconds.

```
In [4]: sample_df.head()
```

Out[4]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	

- In the data preprocessing phase the input N of the store_and_fwd_flag was converted to numeric value 0. conversions of parameters happen in this phase.

```
In [6]: #Convert character variables to numeric
        f = lambda x: 0 if x == 'N' else 1

        sample_df["store_and_fwd_flag"] = sample_df["store_and_fwd_flag"].apply(lambda x: f(x))
```

```
In [7]: #Check result
        sample_df["store_and_fwd_flag"].value_counts()
```

```
Out[7]: 0    1450599
        1       8045
        Name: store_and_fwd_flag, dtype: int64
```

- The training of the XGBoost model takes place on train.csv in Modelling phase. Here our model tries to minimize the eval-rmse and train-rmse, rmse stands for *root mean square error*

```
In [20]: #Train model
         gbm = xgb.train(params,
                         dtrain,
                         num_boost_round = nrounds,
                         evals = watchlist,
                         verbose_eval = True
                         )
```

```
[0]     eval-rmse:2.00792       train-rmse:2.00719
[1]     eval-rmse:1.91382       train-rmse:1.91311
[2]     eval-rmse:1.8224        train-rmse:1.82168
[3]     eval-rmse:1.73565       train-rmse:1.73494
[4]     eval-rmse:1.65361       train-rmse:1.6529
```

the last 5 iterations

```
[94]    eval-rmse:0.340338      train-rmse:0.275842
[95]    eval-rmse:0.340181      train-rmse:0.275018
[96]    eval-rmse:0.339961      train-rmse:0.273998
[97]    eval-rmse:0.339777      train-rmse:0.273135
[98]    eval-rmse:0.33961       train-rmse:0.27231
[99]    eval-rmse:0.339479      train-rmse:0.271249
```

- save the model with file named xgb_model.sav

```
In [25]: filename = "xgb_model.sav"
         pickle.dump(gbm, open(filename, 'wb'))
```

The Genetic_algorithm.ipynb code was executed in the following order.

- First all the libraries were imported as mentioned above.

```
In [1]: import pandas as pd
        import numpy as np
        import xgboost as xgb

        from copy import copy
        import datetime
        import pickle
        from geopy.geocoders import Nominatim
```

- The saved model xgb_model.sav is loaded for applying the genetic algorithm on it.

```
In [2]: filename = "xgb_model.sav"

In [3]: loaded_model = pickle.load(open(filename, 'rb'))
```

- Genetic Algorithm pseudo code

```
START
Generate the initial population
Compute fitness
REPEAT
   Selection
   Crossover
   Mutation    //mutation occurs to maintain diversity and
```

```
                    prevent premature convergence
   Compute fitness
UNTIL population has converged
STOP
```

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

```
current_generation = make_children(breeders, children_per_couple=3)
```

```
Generation 0: 500
['L7', 'L11', 'L3', 'L1', 'L2', 'L4', 'L10', 'L9', 'L8', 'L5', 'L6', 'L7']
Generation 5: 525
['L6', 'L7', 'L2', 'L1', 'L11', 'L3', 'L4', 'L8', 'L9', 'L10', 'L5', 'L6']
Generation 10: 525
['L6', 'L7', 'L2', 'L1', 'L11', 'L3', 'L5', 'L10', 'L8', 'L9', 'L4', 'L6']
Generation 15: 525
['L6', 'L7', 'L2', 'L1', 'L11', 'L3', 'L9', 'L8', 'L10', 'L5', 'L4', 'L6']
Generation 20: 525
['L6', 'L7', 'L2', 'L1', 'L11', 'L3', 'L9', 'L8', 'L10', 'L5', 'L4', 'L6']
```

```
current_generation = create_generation(list(test_locations.keys()),population=500)
fitness_tracking, best_guess = evolve_to_solve(current_generation, 100, 150, 70, 0.5, 3, 5, verbose=True)
```

```
Generation 5: 330
Current Best Score:  198.98142337799072
['L9', 'L7', 'L6', 'L4', 'L1', 'L2', 'L11', 'L3', 'L5', 'L10', 'L8', 'L9']
Generation 10: 330
Current Best Score:  201.57816219329834
['L10', 'L8', 'L11', 'L9', 'L1', 'L2', 'L7', 'L6', 'L4', 'L3', 'L5', 'L10']
Generation 15: 330
Current Best Score:  202.5793581008911
['L10', 'L8', 'L11', 'L9', 'L1', 'L2', 'L7', 'L6', 'L4', 'L3', 'L5', 'L10']
Generation 20: 330
Current Best Score:  202.5793581008911
['L10', 'L8', 'L11', 'L9', 'L1', 'L2', 'L7', 'L6', 'L4', 'L3', 'L5', 'L10']
```

Conclusions:

- We observe here that the next generation's fitness score isn't incrementing anymore and the current best score remains the same, hence we have reached the optimal path that we have to follow. **So we can conclude that for given points L1 to L11, the path we should follow for minimizing the total time(i.e., f(x)) is ['L10', 'L8', 'L11', 'L9', 'L1', 'L2', 'L7', 'L6', 'L4', 'L3', 'L5', 'L10'].**

- In general we can conclude that our model takes significantly less time than other existing algorithms.
- Advantages of using Genetic Algorithm is that it is much faster as compared to traditional algorithm and also gives the accurate path. The result is not guaranteed to be the best solution possible, but it should be close enough. There is little tradeoff between time and accuracy but it's worth it.

Future Works

Route planning would be the next step for this project. For instance, it is possible to incorporate Google Maps API and plan out the exact pathing between each pair of points. Also, the genetic algorithm assumes static time of the day. Accounting for time of the day, while important, is a much more complex problem to solve, and it might require a different approach to constructing the input space altogether.

References:

- XGBoost: A Scalable Tree Boosting System- Tianqi Chen, Carlos Guestrin
- XGboost Wikipedia: https://en.wikipedia.org/wiki/XGBoost
- Genetic Algorithm Wikipedia: https://en.wikipedia.org/wiki/Genetic_algorithm
- A High-Performance Genetic Algorithm: Using Traveling Salesman Problem as a Case- Chun-Wei Tsai, Shih-Pang Tseng, Ming-Chao Chiang, Chu-Sing Yang, and Tzung-Pei Hong
- Genetic Algorithm for Optimizing Routing Design and Fleet Allocation of Freeway Service Overlapping Patrol – Xiuqiao Sun, Jian Wang, Weitiao Wu and Wenjia Liu

Note:

Github repository link for this project
https://github.com/rbnhd/Travel-Time-Optimization-using-Machine-Learning