Route Optimization and Visualization for Sales Vehicles

Team - Aegisrockers

- Veeramanikandan M *Ph.D*
- Ankur Anand
- Pramod Krishna
- Praveen

Input

1. Distribution Point Parameters - Lat-long Input for multiple Distribution points, Number of Vehicle, Type of Vehicle, Vehicle Identifier

2. Vehicle Parameters - Type of Vehicle, Cost per Km, Avg Speed, Maximum distance

that in a day, Maximum hours to return to Distribution Point

3. Delivery Points – Lat-Long of Delivery Point, Potential Sales, Number of minimum visits in a week

4. Distance – Ability to pick up the distance between Lat-Long from Google Distance API

Work Overview

- Forecast the need of the each client on weekly bases and aggregate the total need of the each store.
- Machine Learning Model will be used for this forecasting
- Vehicle Routing Problem with Time Windows (VRPTW) for delivering this items to the stores with constraints like number of vehicles , vehicles capacity and operation cost.

Work Overview (cntd.,)

- Vehicle Routing Problem with Time Windows (VRPTW) for delivering this items to the stores with constraints like number of vehicles , vehicles capacity and operation cost.
- Based the demand and vehicles availability the optimized route us calculated using VRPTW as per https://doi.org/10.1016/j.jksus.2010.03.002
- Plot the optimized route for the each vehicle on the google map
- Total operation cost and optimized route is displayed in the dashboard

Modules

- Module 1: Demand Forecasting
- Module 2: Route Optimization

Module 1 : Demand Forecasting

- Used the public dataset from the kaggle
  - https://www.kaggle.com/c/competitive-data-science-predict-future-sales/data
- This dataset contain
  - 10 stores
  - 50 items
- We developed the Xgboost Regression to predict the sales of the particular store

Module 1 : Demand Forecasting

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle
import time
from xgboost import XGBRegressor
```

```
/home/raju/anaconda3/lib/python3.7/site-packages/distributed,
ble IP address for reaching '8.8.8.8', defaulting to '127.0.
  RuntimeWarning,
```

```
train = pd.read_csv('input/train.csv', parse_dates=['date'])
train=train.set_index('date')
train.head()
```

date	store	item	sales
2013-01-01	1	1	13
2013-01-02	1	1	11
2013-01-03	1	1	14
2013-01-04	1	1	13
2013-01-05	1	1	10

```
train['day'] = train.index.day
train['month'] = train.index.month
train['year'] = train.index.year
```

```
train.head()
```

date	store	item	sales	day	month	year
2013-01-01	1	1	13	1	1	2013
2013-01-02	1	1	11	2	1	2013
2013-01-03	1	1	14	3	1	2013
2013-01-04	1	1	13	4	1	2013
2013-01-05	1	1	10	5	1	2013

```
X=train[['store','item','day','month','year']]
Y=train[['sales']].to_numpy()
```

Module 1 : Demand Forecasting

```python
ts = time.time()

model = XGBRegressor(
    max_depth=10,
    n_estimators=1000,
    min_child_weight=0.5,
    colsample_bytree=0.8,
    subsample=0.8,
    eta=0.1,
#     tree_method='gpu_hist',
    seed=42)

model.fit(
    X_train,
    Y_train,
    eval_metric="rmse",
    eval_set=[(X_train, Y_train), (X_valid, Y_valid)],
    verbose=True,
    early_stopping_rounds = 20)

time.time() - ts
```

```
[526]   validation_0-rmse:7.17479       validation_1-rmse:8.00144
[527]   validation_0-rmse:7.17389       validation_1-rmse:8.00181
```

```python
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

```
63.98911589528639
```

```python
X_valid.shape
```

```
(365200, 5)
```

```python
x=np.array([[6,35,19,5,2015]])
```

```python
x.shape
```

```
(1, 5)
```

```python
z=model.predict(x)
```

```python
z[0]
```

```
51.85673
```

```python
from joblib import dump, load
dump(model,'model_xg.joblib')
```

```
['model_xg.joblib']
```

Module 1 : Demand Forecasting

127.0.0.1:5000/predict/2/10/31/03/2020

Prediction:100.03445

```python
from flask import Flask, url_for
from distutils.util import strtobool
import numpy as np
app = Flask(__name__)
from joblib import dump, load
model = load('model_xg.joblib')
@app.route('/')
def api_root():
    return 'Welcome'

@app.route('/predict/<int:store>/<int:item>/<int:day>/<int:month>/<int:year>')
def api_predict(store,item,day,month,year):
        x=np.array([[store,item,day,month,year]])
        z=model.predict(x)
        return 'Prediction:'+ str(z[0])
if __name__ == '__main__':
    app.run()
```

Module 1 : Demand Forecasting

- Fine tuned the model
- Exported the module
- Created the API service using the trained model

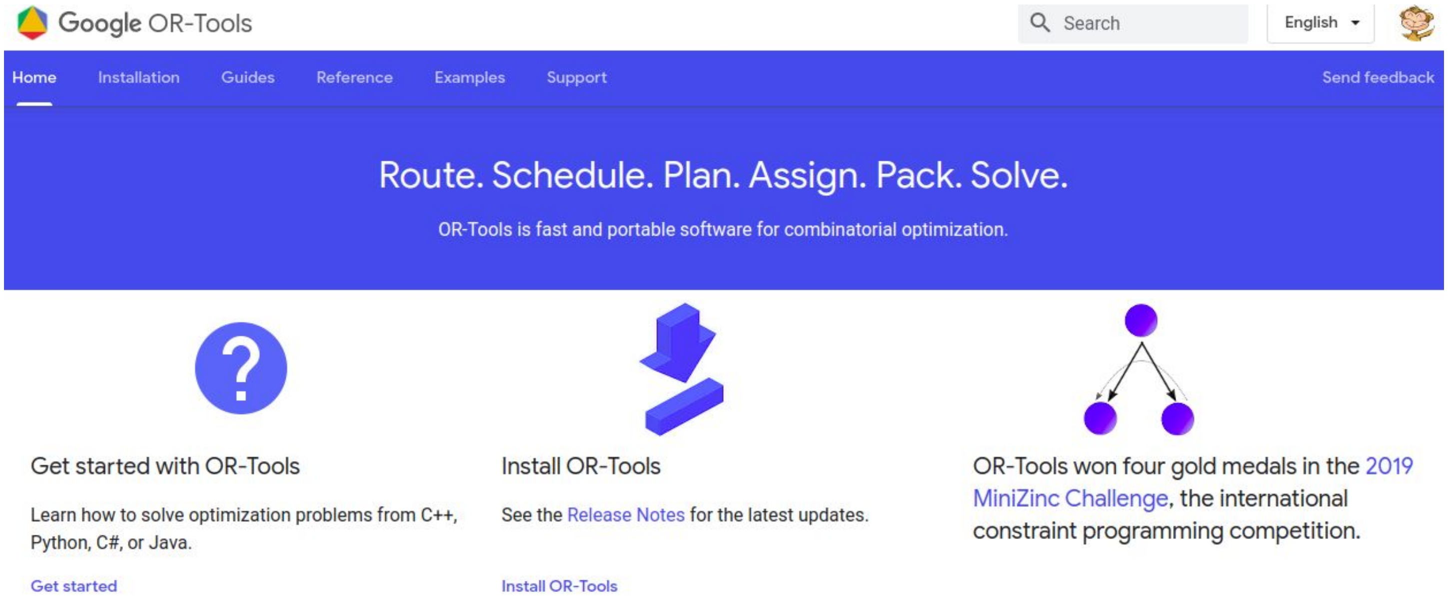
Module 2 : Route Optimization

- The main goals for the vehicle routing problem
  - reducing the vehicle operating cost
  - on time delivery with vehicle capacity
  - Converting all the stores
  - Less waiting time
- To handle this we have used the VRPTW algorithm.

Module 2 : Route Optimization

- ***OR-Tools*** is an open source software suite for optimization, tuned for tackling the world's toughest problems in vehicle routing, flows, integer and linear programming, and constraint programming.
- This tool offers half dozen solvers to solve it: commercial solvers such as Gurobi or CPLEX, or open-source solvers such as SCIP, GLPK, or Google's GLOP and award-winning CP-SAT

Module 2 : Route Optimization



Module 2 : Route Optimization

Functions:

1) Set the geo location for the stores and warehouse
2) Predict the demand of the each store
3) Calculate the Distance matrix
4) Pass the demand and vehicle details to OR Model
5) Get the Route assignment
6) Calculate the Total cost, Total distance and Total Load of the each vehicle

Module 2 : Route Optimization

```
In [1]: from __future__ import print_function
        from ortools.constraint_solver import routing_enums_pb2
        from ortools.constraint_solver import pywrapcp
        import math
        import numpy as np
        import pandas as pd
        import gmaps
        gmaps.configure(api_key='AIzaSyBqkRYADAkseP67pNYN_0xoMTNL_-7hX5w')
```

```
In [2]: ### lat log information
        warehouse= [12.964229, 77.748080] #white field Base
        s1 = [12.977100, 77.626969] # indranagar
        s2 = [13.038176, 77.599675]# hebbal
        s3 = [12.965785, 77.577250]# kr martket
        s4 = [12.915824, 77.626708]# silkboard
        s5 = [12.895550, 77.581681]# jp nagar
        s6 = [12.929295, 77.588614]# jaya nagar
        s7 = [12.973508, 77.612410]# MG road
        s8 = [12.981078, 77.626962]# ulsoor
        s9 = [12.923290, 77.617858]# madiwala
        s10= [12.999115, 77.670351]# tin factory
        loc=[warehouse,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10]
```

```
def distance(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371 # km

    dlat = math.radians(lat2-lat1)
    dlon = math.radians(lon2-lon1)
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = radius * c
    return d

def cal_dis(loc):
    dis_mat=np.array([])
    for i in range(len(loc)):
        temp=np.array([])
        for j in range(len(loc)):
            y=distance(loc[i],loc[j])
            temp=np.append(temp,y)
        dis_mat=np.append(dis_mat, temp,axis=0)
    dis_mat=dis_mat.reshape((11,11))
    return dis_mat
```

```
df1=pd.DataFrame(columns=['ws','s1','s2','s3','s4','s5','s6','s7','s8','s9','s10'
```

Module 2 : Route Optimization

```
df1.head(11)
```

	ws	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10
ws	0.000000	13.201131	18.059341	18.511992	14.211873	19.583922	17.712246	14.737171	13.257029	14.828142	9.272579
s1	13.201131	0.000000	7.407201	5.532390	6.813639	10.310963	6.747716	1.627329	0.442334	6.064312	5.299689
s2	18.059341	7.407201	0.000000	8.408191	13.916676	15.978698	12.166190	7.321927	7.003541	12.925774	8.802944
s3	18.511992	5.532390	8.408191	0.000000	7.719460	7.824525	4.240272	3.905460	5.648665	6.457054	10.746995
s4	14.211873	6.813639	13.916676	7.719460	0.000000	5.375823	4.391912	6.598665	7.255966	1.268541	10.399155
s5	19.583922	10.310963	15.978698	7.824525	5.375823	0.000000	3.826772	9.286214	10.701677	4.988873	14.998287
s6	17.712246	6.747716	12.166190	4.240272	4.391912	3.826772	0.000000	5.551506	7.100940	3.238954	11.778016
s7	14.737171	1.627329	7.321927	3.905460	6.598665	9.286214	5.551506	0.000000	1.787393	5.615110	6.893500
s8	13.257029	0.442334	7.003541	5.648665	7.255966	10.701677	7.100940	1.787393	0.000000	6.501026	5.111117
s9	14.828142	6.064312	12.925774	6.457054	1.268541	4.988873	3.238954	5.615110	6.501026	0.000000	10.170735
s10	9.272579	5.299689	8.802944	10.746995	10.399155	14.998287	11.778016	6.893500	5.111117	10.170735	0.000000

```
def create_data_model(dis_mat,demand):
    """Stores the data for the problem."""
    data = {}
    data['distance_matrix'] = dis_mat
    data['demands'] = demand
    data['vehicle_capacities'] = [10, 5, 10, 15]
    data['num_vehicles'] = 4
    data['cost']=[10,10,10,10]
    data['depot'] = 0
    return data
```

Module 2 : Route Optimization

```
#load ML model
from joblib import dump, load
from datetime import date
model = load('model_xg.joblib')

#demand calculations
stores=10
items=50
store_sales=np.array([])
for i in range(10):
    temp_total=[]
    for j in range(50):
        day=date.today().day+5
        month=date.today().month
        year=date.today().year
        x=np.array([[i,j,day,month,year]])
        z=model.predict(x)
        #print(z)
        temp_total.append(z)
    store_sales=np.append(store_sales,temp_total)
demand_all=store_sales.reshape(10,50)
#demand_all.shape
de=np.array(np.sum(demand_all,axis=1,dtype=int).tolist())
de=np.append(0,de)/1000
demand=de.tolist()
```

```
demand
```

```
[0.0, 2.488, 2.488, 3.494, 3.225, 2.75, 2.017, 2.114, 1.928, 3.268, 2.874]
```

Module 2 : Route Optimization

```
def print_solution(data, manager, routing, assignment):
    """Prints assignment on console."""
    total_distance = 0
    total_load = 0
    total_cost = 0
    routes=[]
    for vehicle_id in range(data['num_vehicles']):
        index = routing.Start(vehicle_id)
        plan_output = 'Route for vehicle {}:\n'.format(vehicle_id)
        route_distance = 0
        route_load = 0
        v_route=np.array([])
        while not routing.IsEnd(index):
            node_index = manager.IndexToNode(index)
            route_load += data['demands'][node_index]
            plan_output += ' {0} Load({1}) -> '.format(node_index, route_load)
            v_route=np.append(v_route,node_index)
            previous_index = index
            index = assignment.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(previous_index, index, vehicle_id)
        plan_output += ' {0} Load({1})\n'.format(manager.IndexToNode(index),route_load)
        v_route=np.append(v_route,manager.IndexToNode(index))
        plan_output += 'Distance of the route: {}m\n'.format(route_distance)
        plan_output += 'Load of the route: {}\n'.format(route_load)
        route_cost=route_distance*data['cost'][vehicle_id]
        plan_output += 'Cost of the route: {}\n'.format(route_cost)
        #print(v_route)
        routes.append([v_route])
        print(plan_output)
        total_distance += route_distance
        total_load += route_load
        total_cost += route_cost
    print('Total distance of all routes: {}m'.format(total_distance))
    print('Total load of all routes: {}'.format(total_load))
    print('Total Cost of all routes: {}'.format(total_cost))
    return routes
```

Module 2 : Route Optimization

```
data=create_data_model(dis,demand)
manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
                                       data['num_vehicles'], data['depot'])
routing = pywrapcp.RoutingModel(manager)
transit_callback_index = routing.RegisterTransitCallback(distance_callback)
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
demand_callback_index = routing.RegisterUnaryTransitCallback(demand_callback)
routing.AddDimensionWithVehicleCapacity(demand_callback_index,
    0,  # null capacity slack
    data['vehicle_capacities'],  # vehicle maximum capacities
    True,  # start cumul to
    'Capacity')
search_parameters = pywrapc
search_parameters.first_sol
assignment = routing.SolveW

if assignment:
    y=print_solution(data, r
```

```
Route for vehicle 0:
 0 Load(0.0) ->  0 Load(0.0)
Distance of the route: 0m
Load of the route: 0.0
Cost of the route: 0

Route for vehicle 1:
 0 Load(0.0) ->  0 Load(0.0)
Distance of the route: 0m
Load of the route: 0.0
Cost of the route: 0

Route for vehicle 2:
 0 Load(0.0) ->  8 Load(1.928) ->  1 Load(4.416) ->  2 Load(6.904) ->  10 Load(9.778) ->  0 Load(9.778)
Distance of the route: 37m
Load of the route: 9.778
Cost of the route: 370

Route for vehicle 3:
 0 Load(0.0) ->  4 Load(3.225) ->  9 Load(6.493) ->  5 Load(9.243) ->  6 Load(11.26) ->  3 Load(14.754) ->  7 Load
(16.868) ->  0 Load(16.868)
Distance of the route: 43m
Load of the route: 16.868
Cost of the route: 430

Total distance of all routes: 80m
Total load of all routes: 26.646
Total Cost of all routes: 800
```

Module 2 : Route Optimization

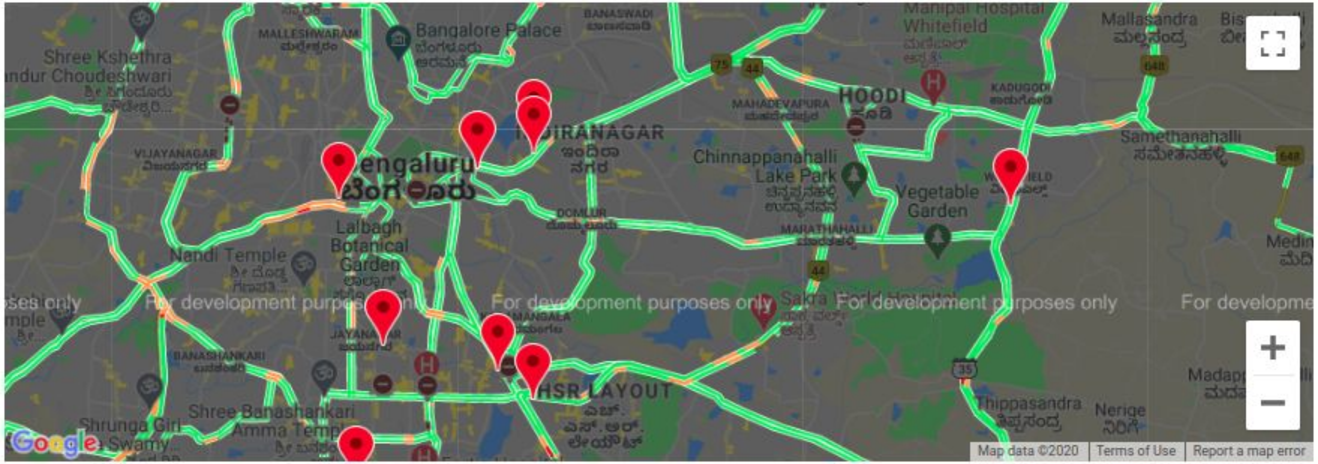
```
for i in range(data['num_vehicles']):
    color=['orange','red','blue','green']
    for j in y[i]:
        #print("\n Vehile ",i," delivery the ",j.shape[0]," stores")
        print("drawing{} = gmaps.drawing_layer(features=[".format(i))
        prev=0
        for k in j:
            #print(loc[int(k)])
           # print(loc[int(k)],",",loc[int(prev)],color[i])
            print("gmaps.Line(",loc[int(k)],",",loc[int(k)],",","stroke_weight=5.0,stroke_color='{}'),".format(color
            prev=k
        print("])")
        print("fig.add_layer(drawing{})".format(i))
print("fig")
```

```
drawing0 = gmaps.drawing_layer(features=[
gmaps.Line( [12.964229, 77.74808] , [12.964229, 77.74808] , stroke_weight=5.0,stroke_color='orange'),
gmaps.Line( [12.964229, 77.74808] , [12.964229, 77.74808] , stroke_weight=5.0,stroke_color='orange'),
])
fig.add_layer(drawing0)
drawing1 = gmaps.drawing_layer(features=[
gmaps.Line( [12.964229, 77.74808] , [12.964229, 77.74808] , stroke_weight=5.0,stroke_color='red'),
gmaps.Line( [12.964229, 77.74808] , [12.964229, 77.74808] , stroke_weight=5.0,stroke_color='red'),
])
fig.add_layer(drawing1)
drawing2 = gmaps.drawing_layer(features=[
gmaps.Line( [12.964229, 77.74808] , [12.964229, 77.74808] , stroke_weight=5.0,stroke_color='blue'),
gmaps.Line( [12.981078, 77.626962] , [12.981078, 77.626962] , stroke_weight=5.0,stroke_color='blue'),
gmaps.Line( [12.9771, 77.626969] , [12.9771, 77.626969] , stroke_weight=5.0,stroke_color='blue'),
gmaps.Line( [13.038176, 77.599675] , [13.038176, 77.599675] , stroke_weight=5.0,stroke_color='blue'),
gmaps.Line( [12.999115, 77.670351] , [12.999115, 77.670351] , stroke_weight=5.0,stroke_color='blue'),
gmaps.Line( [12.964229, 77.74808] , [12.964229, 77.74808] , stroke_weight=5.0,stroke_color='blue'),
])
```

Module 2 : Route Optimization

```
s6= gmaps.Marker(loc[5], info_box_content='Store6')
s7= gmaps.Marker(loc[6], info_box_content='Store7')
s8= gmaps.Marker(loc[7], info_box_content='Store8')
s9= gmaps.Marker(loc[8], info_box_content='Store9')
s10= gmaps.Marker(loc[9], info_box_content='Store10')
drawing = gmaps.drawing_layer(features=[ws,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10])
fig.add_layer(drawing)
fig.add_layer(gmaps.traffic_layer())
fig
```

```
[directions layer] You are not allowed to use Google's directions service
```



Module 2 : Route Optimization

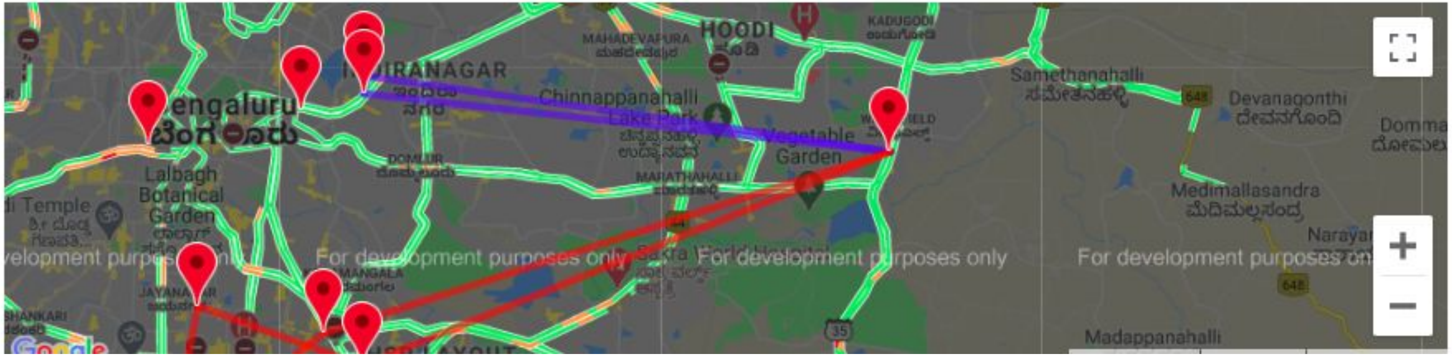
```
l8 = gmaps.Line(loc[9],loc[0],stroke_weight=3.0,stroke_color='red')
drawing1 = gmaps.drawing_layer(features=[l4,l5,l6,l7,l8])
fig.add_layer(drawing1)


fig
```

```
[directions layer] You are not allowed to use Google's directions service
```

```
[directions layer] You are not allowed to use Google's directions service
```

```
[directions layer] You are not allowed to use Google's directions service
```



Future Work

- Using RNN-LSTM for prediction with some different large dataset
- Using the Google Direction API, it is not a open source one so we have not used now. We planned to use those for next round.
- Improved fully function dashboard and mobile app with tracking for all end user.
- Including the real time stream processing.