



Snap to Road Functionality for Off-Road Vehicles

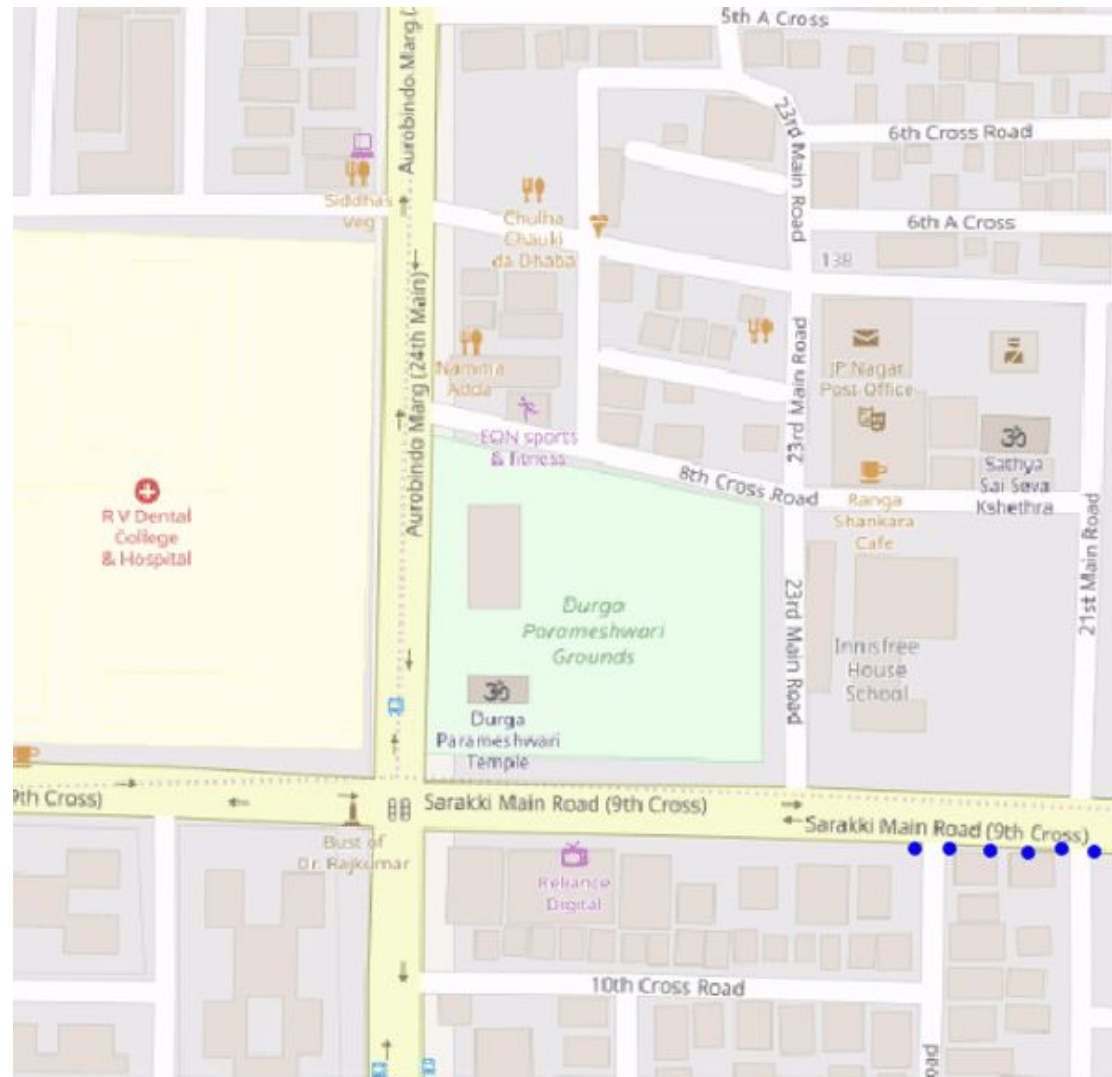
Srivathsan Balaji
Dinesh C
Rahul Gunaseelan
Srinivasan K

Problem Statement

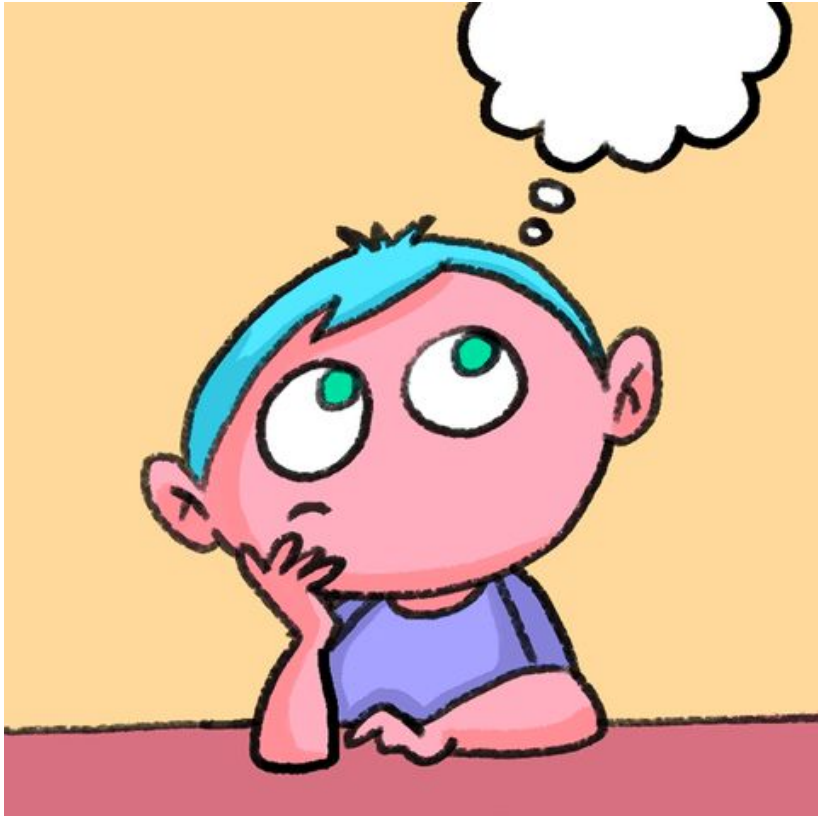
A truck travelling on a Haul Road inside a Jobsite (a quarry, road construction site, mine site, etc.) will be sending GPS data back to Back Office. This GPS data that is based on GNSS satellites will typically have **accuracy issues** based on the visibility of the GNSS satellites to the GPS receiver. We would need capabilities to snap the GPS points back to the Haul Road.

A solution to this problem is to **snap each point to the nearest road segment**. Need a post-processing engine to align the GPS data on a lane of the road. This needs to be near real-time.

This is what we should do but inside a mine!?



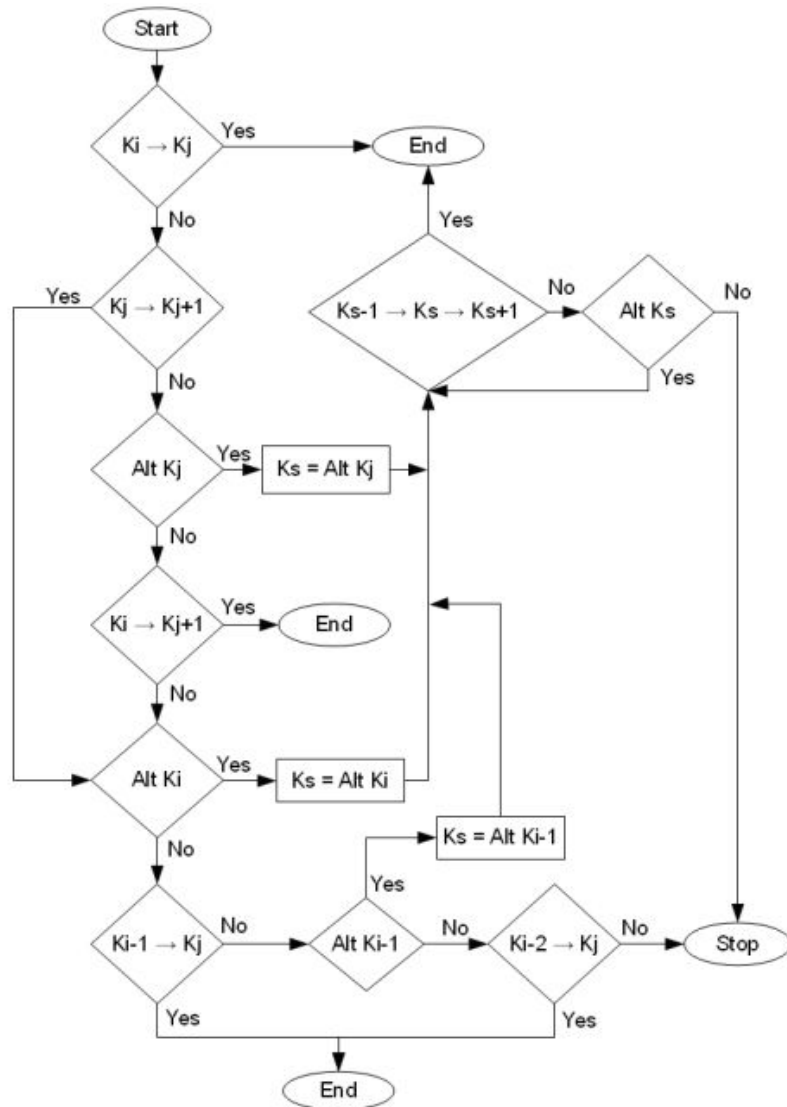
But how do we do it?



- ❖ Machine Learning
- ❖ Deep Learning
- ❖ CNN
- ❖ RNN
- ❖ Map matching algorithms
- ❖ AI
- ❖ Viterbi algorithm - HMM

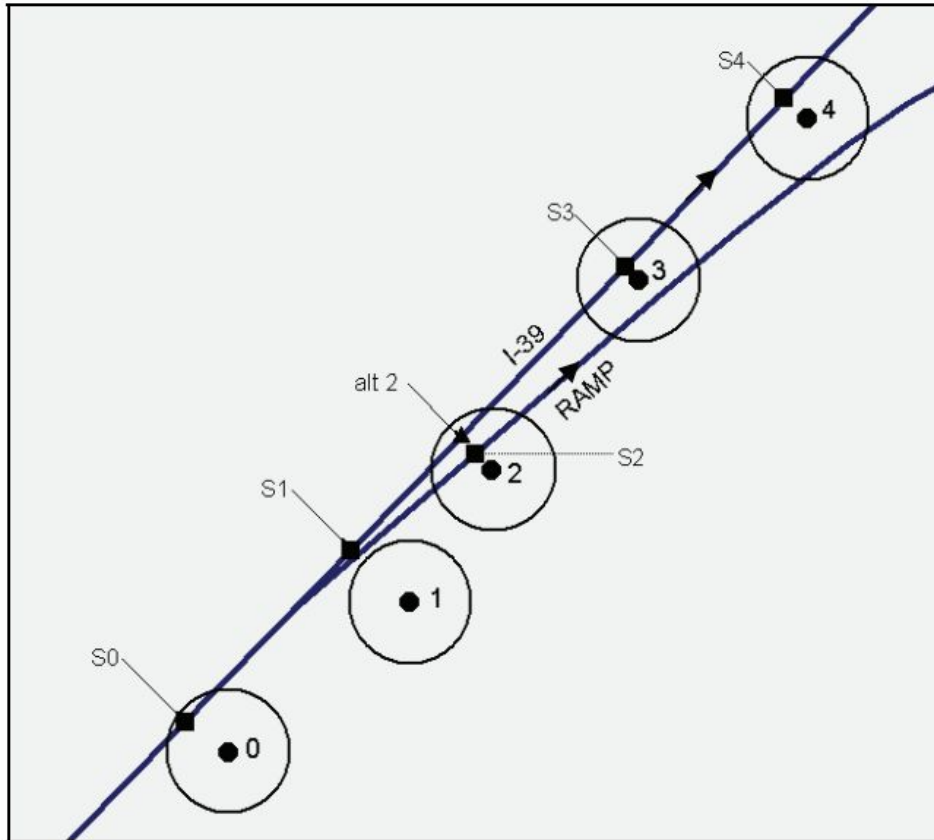
Which to use now?

General Approach - Map Matching Algorithm



This algorithm simply calculates the distance between all the possible points and selected point that we have to approach. The average distance and speed is also considered as a parameters in deciding the snapped point.

Example



Data Points	Shortest Path Distance (ft)	Calculated Speed (mi/h)	Average Recorded Speed (mi/h)	Is Path Feasible?
S0 → S2	392.6	26.8	31.5	YES
S2 → S3	5125.9	699	33	NO
S3 → S4	213	29	35	YES
S0 → alt2	392.8	26.8	31.5	YES
alt2 → S3	215.7	29.4	33	YES

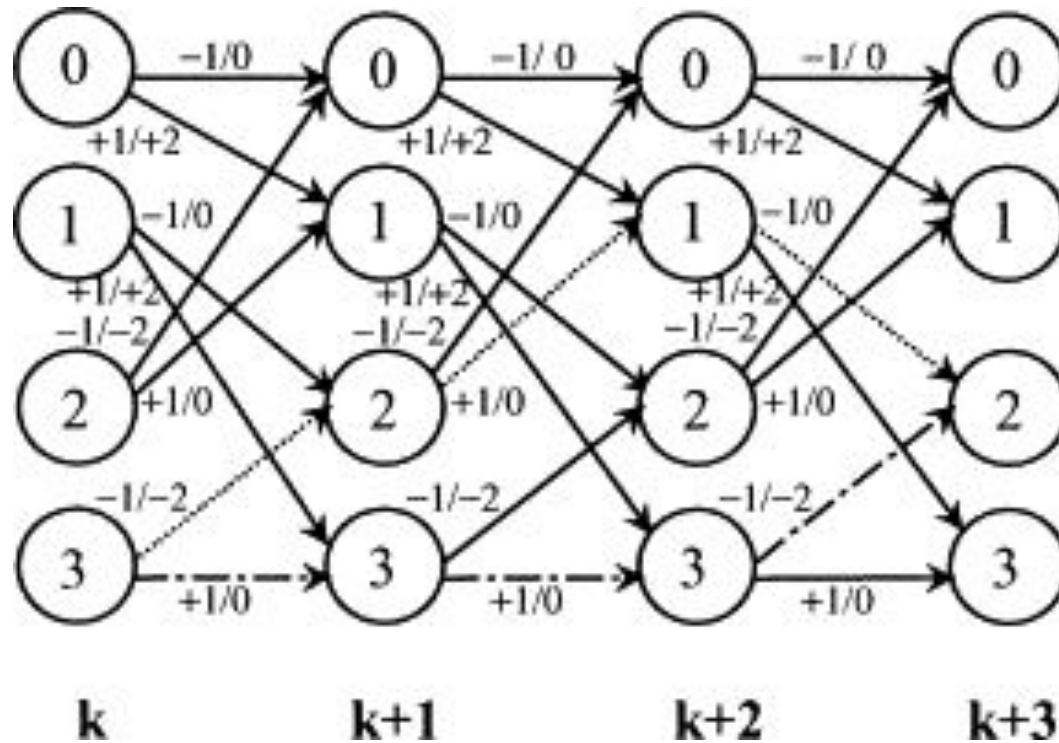
Reference Link: <https://bit.ly/3Pw4llq>

Disadvantages of Map matching Algorithms

- ❖ The key constraints and limitations are the problems associated with **initial identification of vehicle positions, the problem of matching positioning fixes in complex road layouts, performance evaluation, especially in dense urban areas, and development of confidence indicators.**
- ❖ The limitation of just **analysing 4 points and limited computation of distances and speed between the points** arise as a major limitation in the algorithm.
- ❖ So, we tried to move on to other models and algorithms for more **efficiency and accuracy.**

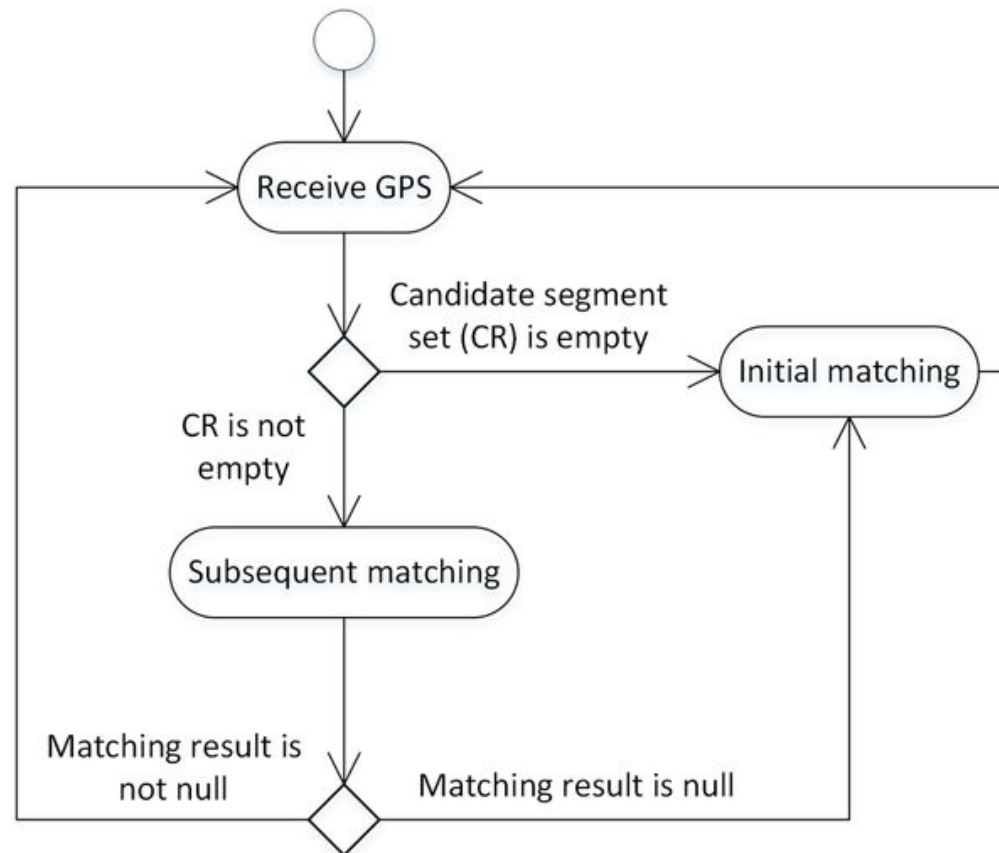
Hidden Markov Model - Viterbi Algorithm

A Hidden Markov model (HMM) is a **statistical Markov model** in which the system being **modeled** is assumed to be a **Markov process**.



Viterbi Algorithm

It is an algorithm that works with a Hidden Markov model that identifies the **most probable sequence of state transitions** for a given observation sequence.

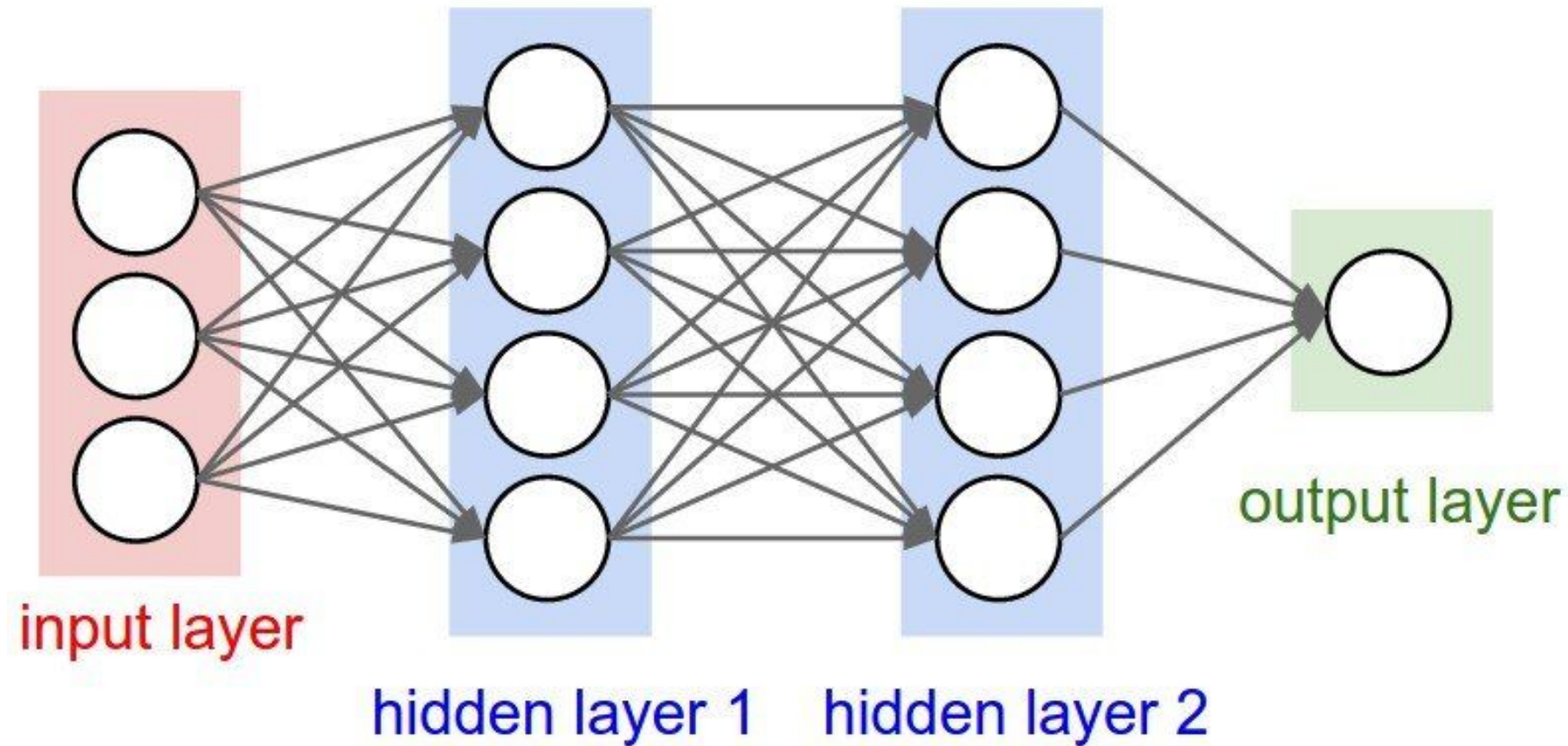


Neural Network Based Approach

- To provide an effective way to map point to nearby point lying in the segment we have **used 3 layer neural networks with 2 hidden layers**.
- The model is used to predict the **logical likelihood of a given GPS data** coordinate being snapped to a particular segment.
- The gps coordinates of the snapped point is determined by **scanning the coordinates of the segment** that it belongs to.
- GitHub link :
<https://github.com/xXAtoZXx/Snap-To-Road>

Neural Network Hidden Layers

The hidden layers would find the correlation of features when the dataset has more features and more sample to work on.

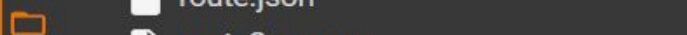




Files



{x}



- ..
- sample_data
- route.json
- route2n_s.csv
- sample(north_south).csv
- sample.csv
- sample2.csv

+ Code + Text



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from random import randint, shuffle
import numpy as np
from sklearn.model_selection import train_test_split as tts
from sklearn import preprocessing
import pandas as pd
```



```
def base_model(n):
    model = Sequential([
        tf.keras.layers.Dense(units = 100,activation = 'relu',input_shape = (2,)),
        tf.keras.layers.Dense(units = 50,activation = 'relu'),
        tf.keras.layers.Dense(units = n,activation = 'softmax')
    ])
    return model
```



```
[ ] def predictsegment(model , testing):
    predictions = model.predict(testing,batch_size = 10, verbose = 1)
    return predictions
```

```
[ ] df = pd.read_csv("sample2.csv")
x = df[['Latitude','Longitude']]
y = df.filter(['Segment'],axis = 1)
x = preprocessing.StandardScaler().fit(x).transform(x)
x_train, x_test, y_train, y_test = tts(x,y,test_size = 0.2,random_state = 10)
x_test1 = x_test
```





+ Code + Text

 print(x)

```
[[ -0.40322782 -0.17902528]
 [ -0.40322782 -0.15344812]
 [ -0.40322782 -0.12787097]
 ...
 [ 0.3998211 -0.01677459]
 [ 0.3998211 0.00880256]
 [ 0.3998211 0.03437972]]
```

```
[ ] model = base_model(6)
    df.head()
    # print(x)

    # print(y)
    # give input as number of output segments in each route
```

	Latitude	Longitude	Segment
0	-111.119192	31.828087	0
1	-111.119192	31.828097	0
2	-111.119192	31.828107	0
3	-111.119192	31.828117	0
4	-111.119192	31.828127	0

```
[ ] model.compile(optimizer = Adam(learning_rate = 0.001), loss = "sparse_categorical_crossentropy", metrics = ['accuracy'])
    history = model.fit(x_train,y_train, batch_size = 10,epochs = 25,verbose = 1,validation_split=0.2)
    model.summary()
```


Radius = 6371e3

```
def distanceBetween(point1, point2):
    phi1 = point1[0] * np.pi / 180
    phi2 = point2[0] * np.pi / 180
    deltaPhi = (point2[0] - point1[0]) * np.pi / 180
    deltaLambda = (point2[1] - point1[1]) * np.pi / 180

    a = np.sin(deltaPhi/2) * np.sin(deltaPhi/2) + np.cos(phi1) * np.cos(phi2) * np.sin(deltaLambda/2) * np.sin(deltaLambda/2)

    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))

    return Radius * c

def findMatchedPoint(point, segment):
    file = open('/home/route.json', 'r')
    FileData = json.load(file)
    coordinates = []
    for feature in FileData['features']:
        if feature['properties']['segment'] == segment:
            coordinates.extend(feature['geometry']['coordinates'])
            break
    min = distanceBetween(coordinates[0], point)
    minIndex = 0
    for idx in range(1, len(coordinates)):
        dist = distanceBetween(coordinates[idx], point)
        if (dist < min):
            min = dist
            minIndex = idx
    return coordinates[minIndex]
```



+ Code + Text

Connect

```
[ ] Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 100)	300
dense_1 (Dense)	(None, 50)	5050

```
▶ score = model.evaluate(x_test, y_test, batch_size = 10, verbose =1)
  predictions = predictsegment(model , x_test)
  # y_test.head(100)
  arrpoints = predictions

  # print(predictions)
  print("Test Loss : " , score[0])
  print("Test Accuracy : ", score[1])
  print(score)
```

```
320/320 [=====] - 0s 1ms/step - loss: 0.0791 - accuracy: 0.9633
320/320 [=====] - 0s 1ms/step
Test Loss : 0.07911230623722076
Test Accuracy : 0.9633458852767944
[0.07911230623722076, 0.9633458852767944]
```

```
[ ] df['Segment'].value_counts()
```

```
5    6496
1    3808
2    2016
0    1848
3    1288
4     504
Name: Segment, dtype: int64
```


K-Nearest Neighbors Approach

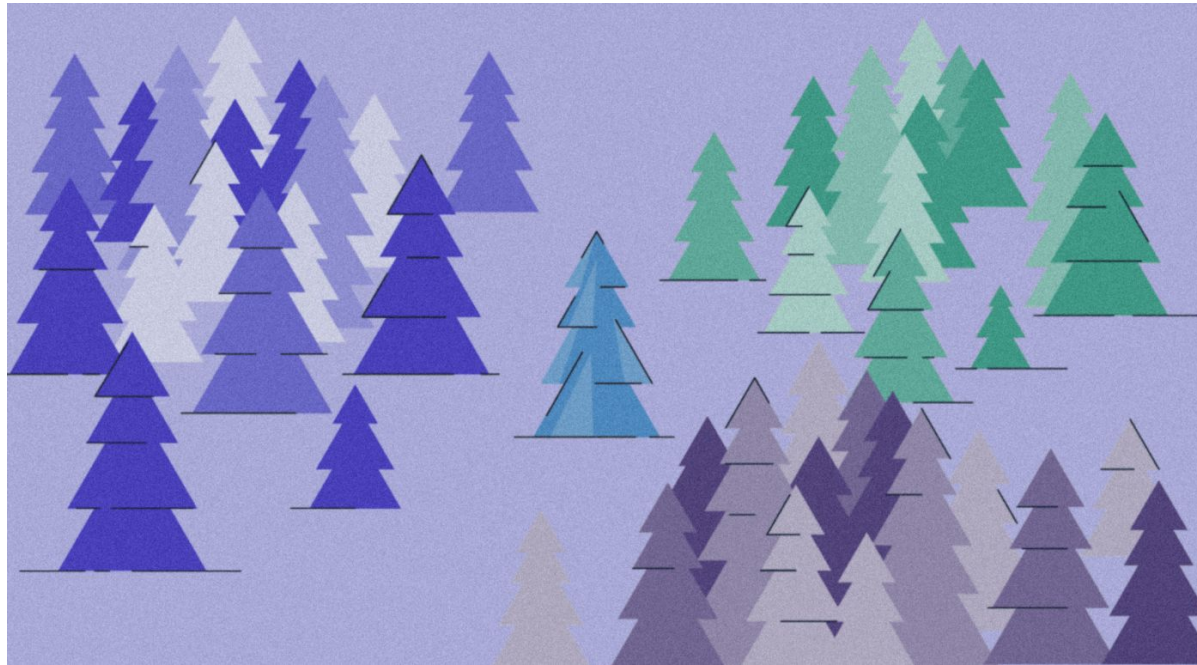
It is one of the Machine learning algorithms based on Supervised Learning.

K-NN algorithm assumes the **similarity between the new case/data and available cases** and put the new case into the category that is most similar to the available categories.

It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category.

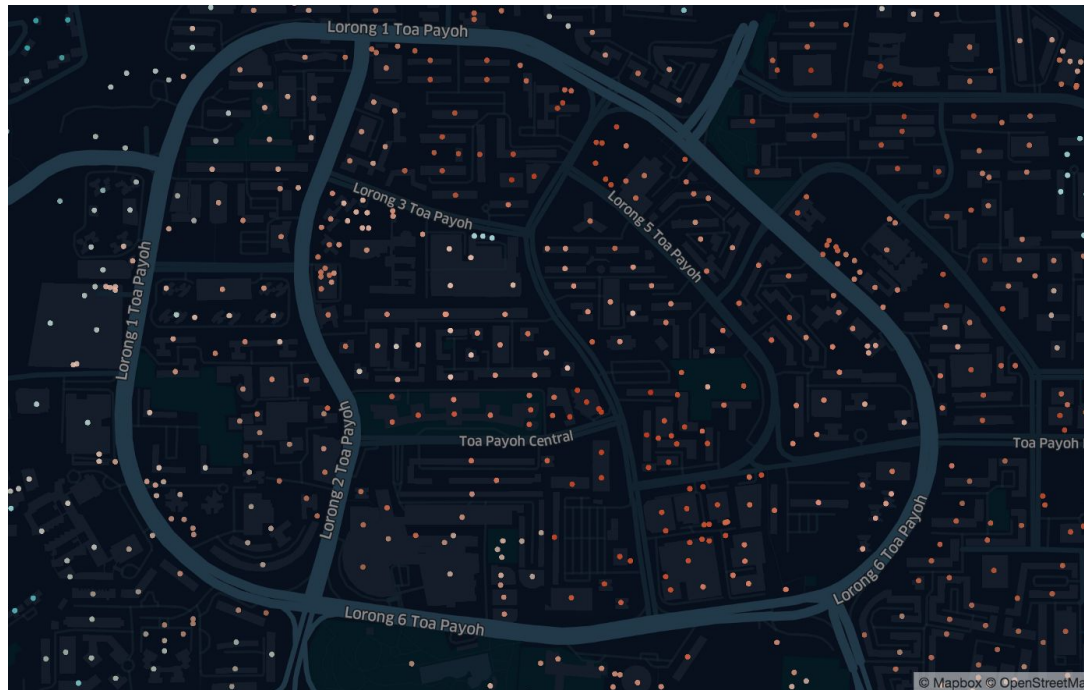


Why do we need KNN Approach ?

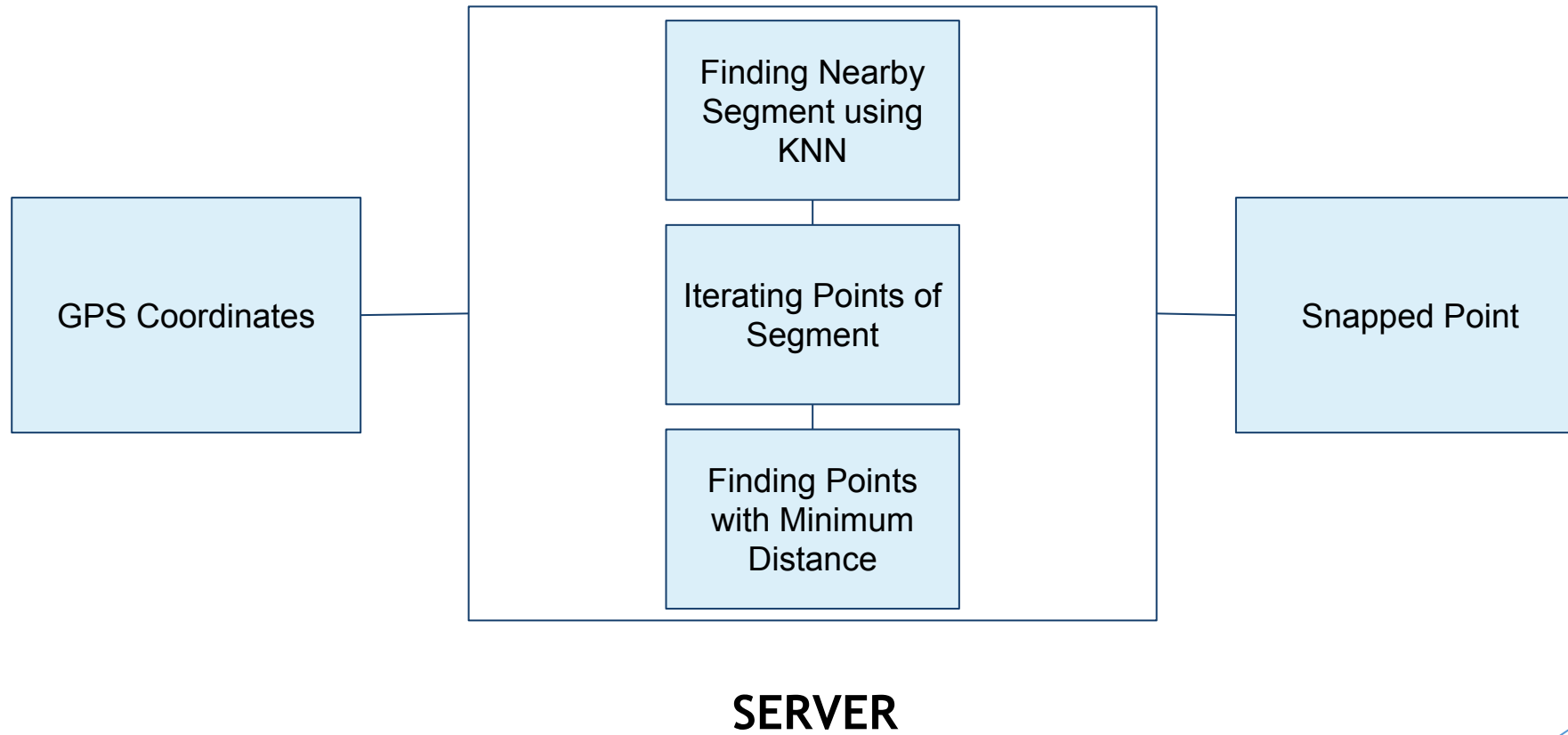


Why do we need KNN Approach ?

Similarly, if we have a scenario where the GPS coordinates could belong to any segment of different routes then we should be able to give the appropriate matched segment. So, we classify the segments into classes and then classify the input data point to respective class using KNN.



Workflow



Sample Implementation of K-NN Approach



model.py > ...

```
1  from sklearn.neighbors import KNeighborsClassifier
2  from sklearn.model_selection import train_test_split
3  import pandas as pd
4  from sklearn.preprocessing import StandardScaler
5
6
7  df = pd.read_csv('sample2.csv')
8
9  X = df.drop(['Segment'], axis=1)
10 y = df['Segment']
11
12
13 scalar = StandardScaler()
14 scalar = scalar.fit(X)
15 X = scalar.transform(X)
16
17
18 X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y, train_size=0.80, random_state=16)
19
20 KNN = KNeighborsClassifier(n_neighbors=50, weights="distance")
21 KNN.fit(X_train, y_train)
22
23
```


server.py > ...

```
1  # Import libraries
2  import numpy as np
3  from flask import Flask, request, jsonify
4  import pickle
5  import pandas as pd
6  import json __name__: str
7  app = Flask(__name__)
8  # Load the model
9  model = pickle.load(open('model.pkl', 'rb'))
10 scalar = pickle.load(open('scalar.pkl', 'rb'))
11
12 Radius = 6371e3
13 def distanceBetween(point1, point2):
14     phi1 = point1[0] * np.pi / 180
15     phi2 = point2[0] * np.pi / 180
16     deltaPhi = (point2[0] - point1[0]) * np.pi / 180
17     deltaLambda = (point2[1] - point1[1]) * np.pi / 180
18     a = np.sin(deltaPhi/2) * np.sin(deltaPhi/2) + np.cos(phi1) * np.cos(phi2) * np.sin(deltaLambda/2) * np.sin(deltaLambda/2)
19     c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
20     return Radius * c
21
22 def findMatchedPoint(point, segment):
23     file = open('route.json', 'r')
24     FileData = json.load(file)
25     coordinates = []
26     for feature in FileData['features']:
27         if feature['properties']['segment'] == segment:
28             coordinates.extend(feature['geometry']['coordinates'])
29             break
30     min = distanceBetween(coordinates[0], point)
```



```
min = distanceBetween(coordinates[0], point)
minIndex = 0
for idx in range(1,len(coordinates)):
    dist = distanceBetween(coordinates[idx], point)
    if(dist < min):
        min = dist
        minIndex = idx
return coordinates[minIndex]
```

```
@app.route('/api',methods=['POST'])
```

```
def predict():
```

```
    # Get the data from the POST request.
```

```
    data = request.get_json(force=True)
```

```
    # Make prediction using model loaded from disk as per the data.
```

```
    print(data)
```

```
    returnList=[]
```

```
    for coords in data['coordinates']:
```

```
        predictVal = pd.DataFrame([coords], columns=['Latitude','Longitude'])
```

```
        predictVal = scalar.transform(predictVal)
```

```
        value = model.predict(predictVal)[0]
```

```
        returnList.append(int(value))
```

```
    # prediction = model.predict([[np.array(data)]])
```

```
    # # Take the first value of prediction
```

```
    # output = prediction
```

```
    # return jsonify(output)
```

```
    print(returnList)
```

```
    return jsonify({"data":returnList})
```

```
# prediction = model.predict([[np.array(data)]])
# # Take the first value of prediction
# output = prediction
# return jsonify(output)
print(returnList)
return jsonify({"data":returnList})
if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

```
1256     "type": "Point",
1257     "coordinates": [
1258         -111.11881524324417,
1259         31.82764256177203
1260     ]
1261   },
1262 },
1263 {
1264   "type": "Feature",
1265   "properties": {},
1266   "geometry": {
1267     "type": "Point",
1268     "coordinates": [
1269       -111.11871600151062,
1270       31.82791830918881
1271     ]
1272   }
1273 },
1274 {
1275   "type": "Feature",
1276   "properties": {},
1277   "geometry": {
1278     "type": "Point",
1279     "coordinates": [
1280       -111.11896008253098,
1281       31.82814392009894
1282     ]
1283   }
1284 }
1285 ]
1286 }
```



POST http://localhost:5000/

No Environment

http://localhost:5000/api

Save

</>

POST http://localhost:5000/api

Send

ParamsAuthHeaders (8)BodyPre-req. Tests Settings

rawJSON

1{
2 "coordinates": [
3 [
4 -111.11881524324417,
5 31.82764256177203
6],
7 [
8 -111.11871600151062,
9 31.82791830918881
10],
11 [
12 -111.11896008253098,
13 31.82814392009894
14]
15]
16 }

Cookies

BodyCookiesHeaders (5)Test Results

200 OK50 ms376 BSave Response

PrettyRawPreviewVisualizeJSON

1{
2 "data": [
3 [
4 -111.11881254997,
5 31.827492199869997
6],
7 [
8 -111.11880387418,
9 31.827839329519996
10],
11 [
12 -111.11894807051,
13 31.828213159819995
14]
15]
16 }

OnlineFind and ReplaceConsole

CookiesCapture requestsBootcampRunnerTrash



```
1253   "type": "Feature",
1254   "properties": {},
1255   "geometry": {
1256     "type": "Point",
1257     "coordinates": [
1258       -111.11881254997,
1259       31.827492199869997
1260     ]
1261   },
1262 },
1263 {
1264   "type": "Feature",
1265   "properties": {},
1266   "geometry": {
1267     "type": "Point",
1268     "coordinates": [
1269       -111.11880387418,
1270       31.827839329519996
1271     ]
1272   },
1273 },
1274 {
1275   "type": "Feature",
1276   "properties": {},
1277   "geometry": {
1278     "type": "Point",
1279     "coordinates": [
1280       -111.11894807851,
1281       31.828213159819995
1282     ]
1283   },
1284 }
```


Problems associated with KNN

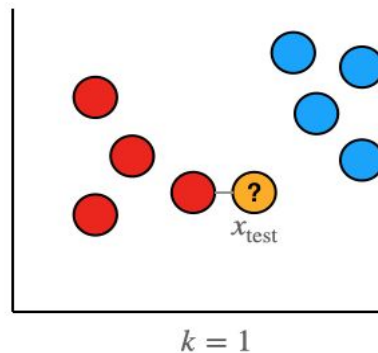
- As the number of data points increase, the model tends to be **overfitted**. Thus, it would become time consuming and inefficient as number of data points increase and as k increases.
- Also, there also exists some coordinates that **overlap with 2 or more segments**, thereby causing ambiguity errors in selection of segments.
- The model would not work with **increasing number of features** and also doesn't work well large datasets.



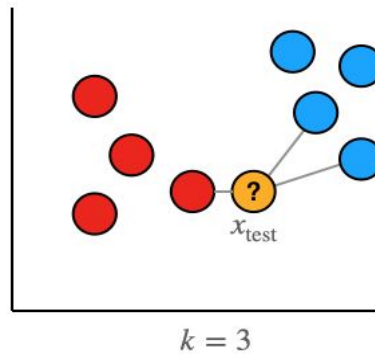
Future Enhancements

Reduction of KNN Span

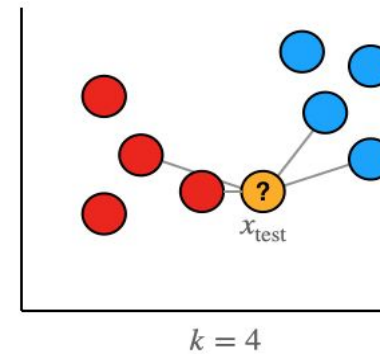
- As the dataset is small our KNN approach would work, however if the data point lies in the overlapping region then it might map to wrong segment.
- In order to minimize the error due to above reason aperture minimization could be implemented.



Nearest point is **red**, so x_{test} classified as **red**



Nearest points are {**red**, **blue**, **blue**} so x_{test} classified as **blue**

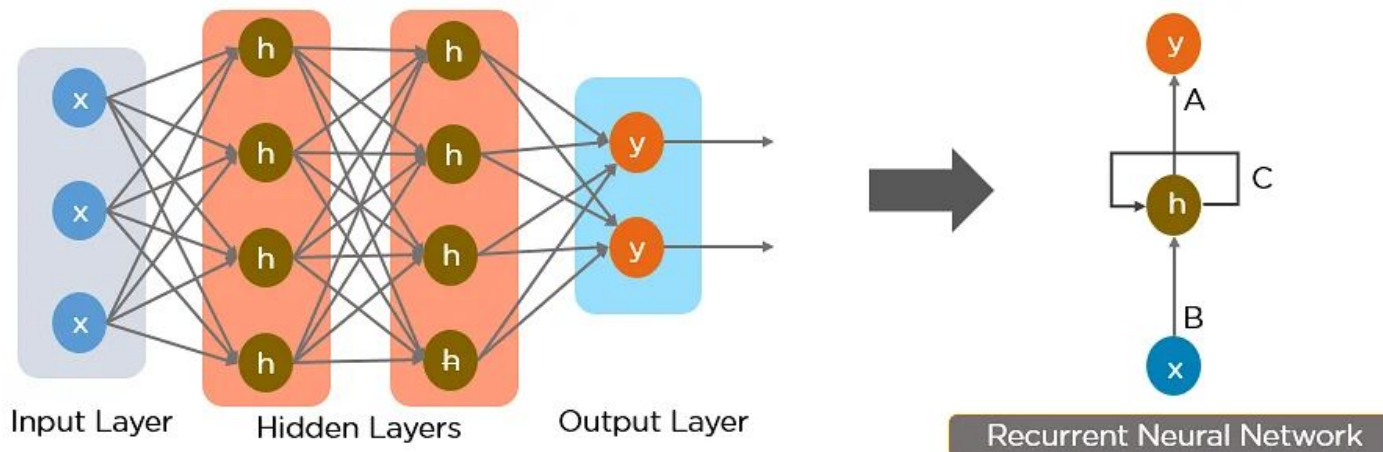


Nearest points are {**red**, **red**, **blue**, **blue**} so classification of x_{test} is not properly defined

Future Enhancements

Recurrent Neural Networks(RNN)

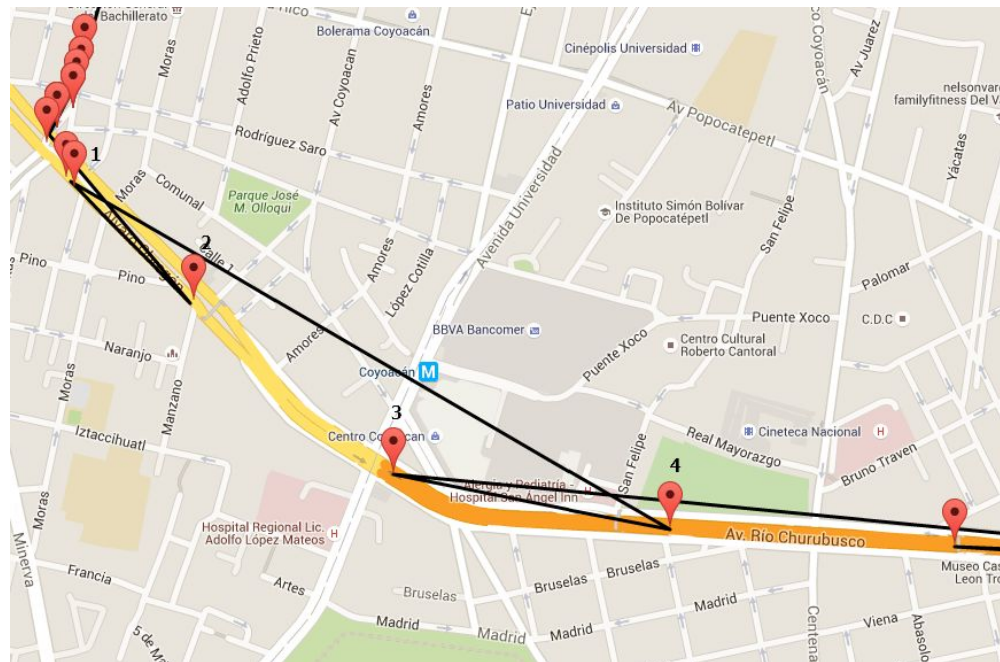
- In a real time scenario, we might have changing roads or there can be **several other factors** affecting the nature of the model. This requires altering the existing model.
- This can be done by using the **Memory capable Nature of the RNN** that use the data obtained from the previous layers and use it to predict the outputs for the new routes and segments without completely retraining the model.



Future Enhancements

Mapping of Snapped Points

- Develop an interface to show the inaccurate and the snapped points on a map basis for visual understanding and implementation.
- Also, improvise our algorithm to find the shortest path to cover the destination.



Future Enhancements

The following data could be added for **optimal selection** of snapping points:

- Elevation
- Machine dimensions
- Track conditions
- Trajectory of points

To Summarize...

- We have considered many of the frequently used algorithms for the given type of the problem statement, and we have concluded that, for the given data set, using KNN with 13 neighbours yielded the best results with an average accuracy of over 90%.
- We also considered the use of Recurrent Neural Networks for more advanced implementations of the same.
- We also suggest the use of RNN due its the dynamic structure and faster runtimes despite its higher pre-processing time and requirement for higher dimensional data.

The background features abstract, overlapping geometric shapes in various shades of blue, primarily on the right side of the frame. These shapes include triangles and polygons of different sizes and opacities, creating a modern, layered effect. The left side of the image is a solid, light blue color.

Thank you!