# Τεχνικές Εξόρυξης Δεδομένων
# Εαρινό Εξάμηνο 2017-2018

## 2η Άσκηση

Γιώργος Μανδηλαράς Α.Μ.: 1115201200097

Θοδωρής Μανδηλαράς Α.Μ.:1115201200098

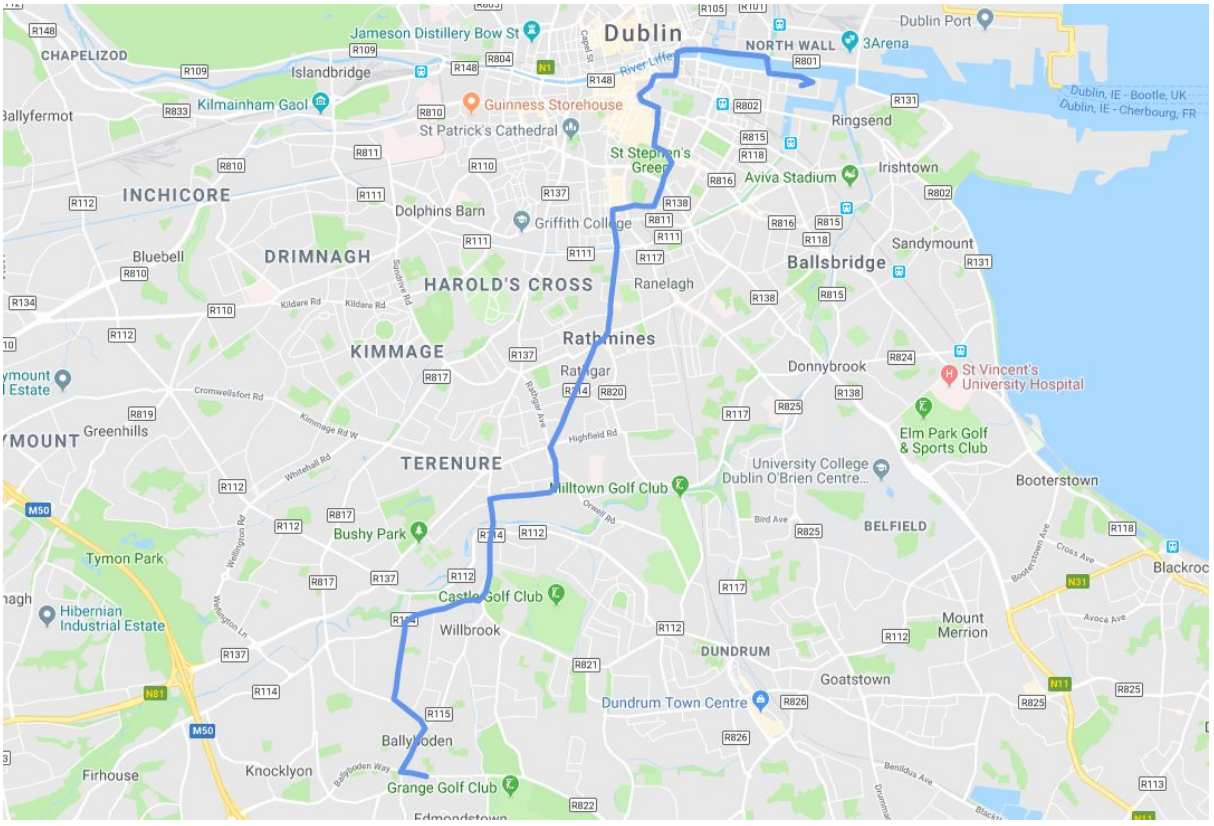
# Ερώτημα 1

## Οπτικοποίηση των Δεδομένων

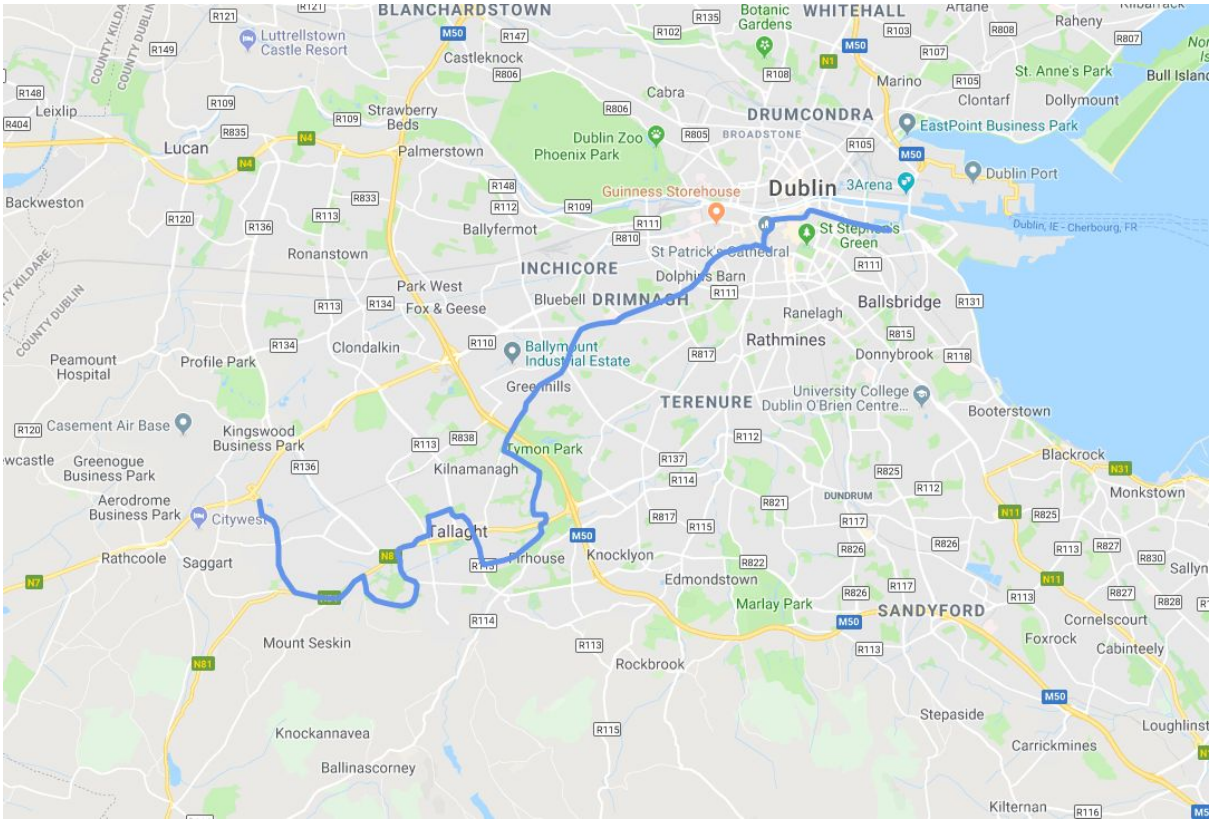
Για αυτό το ερώτημα, στο αρχείο Question_1/Q1_gmplot.py, οπτικοποιήσαμε 5 συγκεκριμένες διαδρομές από διαφορετικές γραμμές λεωφορείων, τις οποίες τα tripId τους είναι τα εξής: 1, 4, 12, 43, 48. Χρησιμοποιήθηκε η βιβλιοθήκη gmplot, και πιο συγκεκριμένα πήραμε τα lats και lons των διαδρομών αυτών, και τα περάσαμε στην συνάρτηση gmplot.GoogleMapPlotter.plot η οποία σχεδίασε τον χάρτη, και με την χρήση της gmplot.GoogleMapPlotter.draw δημιουργήσαμε τα html αρχεία. Τα αρχεία αυτά αποθηκεύονται στο φάκελο Question_1/Q1_produced/ και φέρουν το όνομα του journey Pattern Id τους.

### Αποτελέσματα

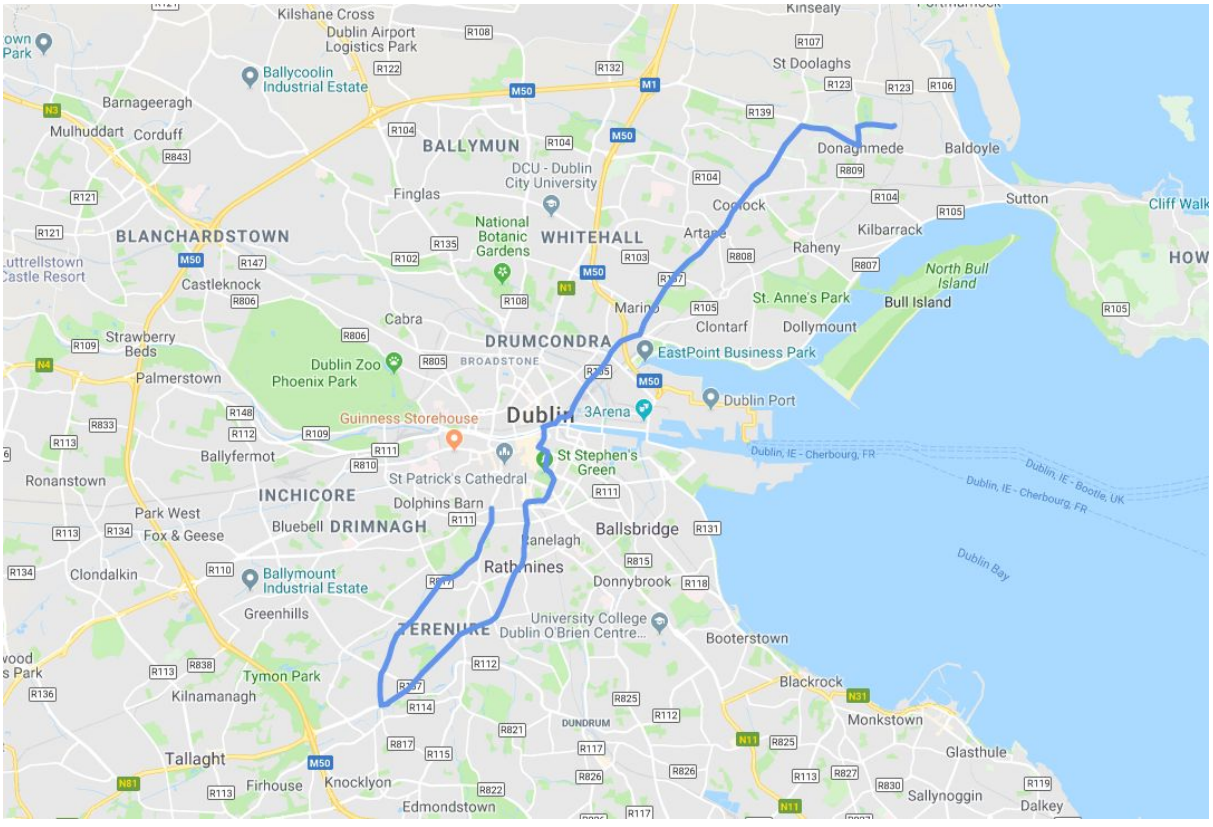
015B1002.html



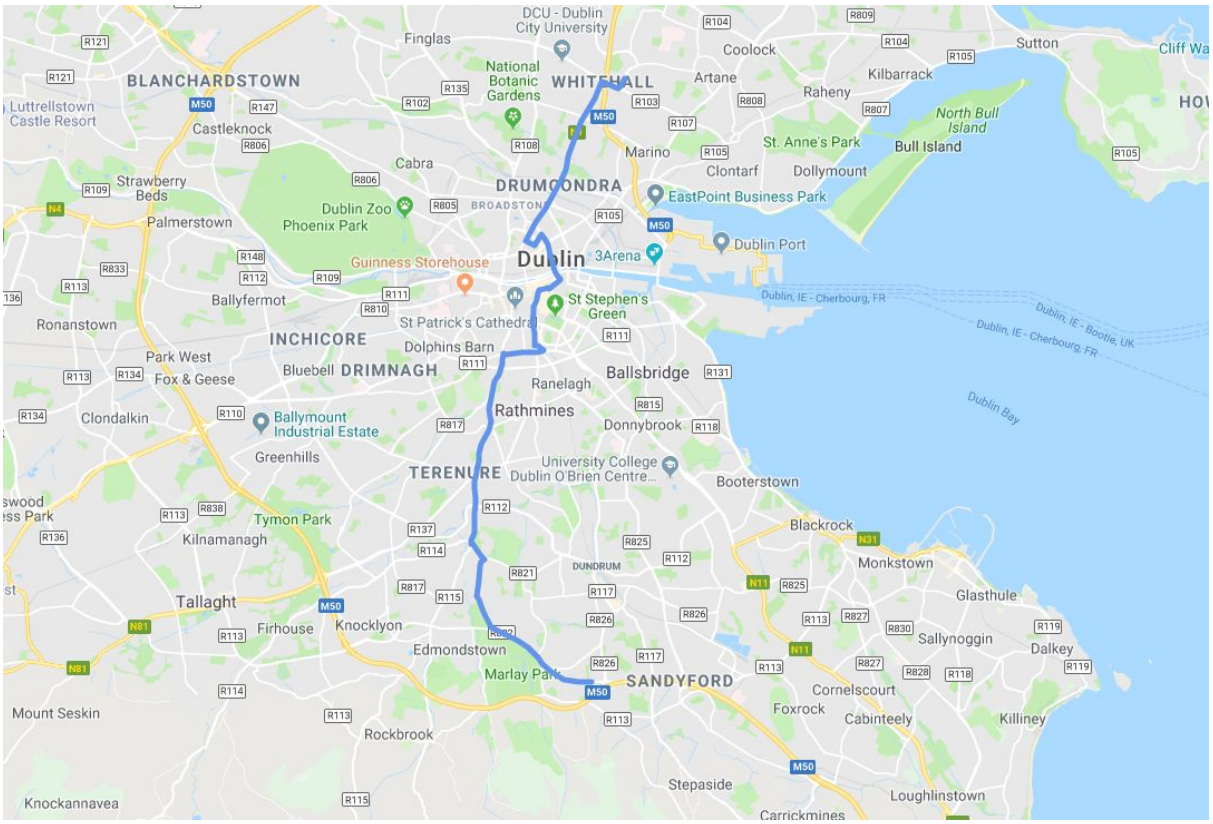
## 077A0001.html



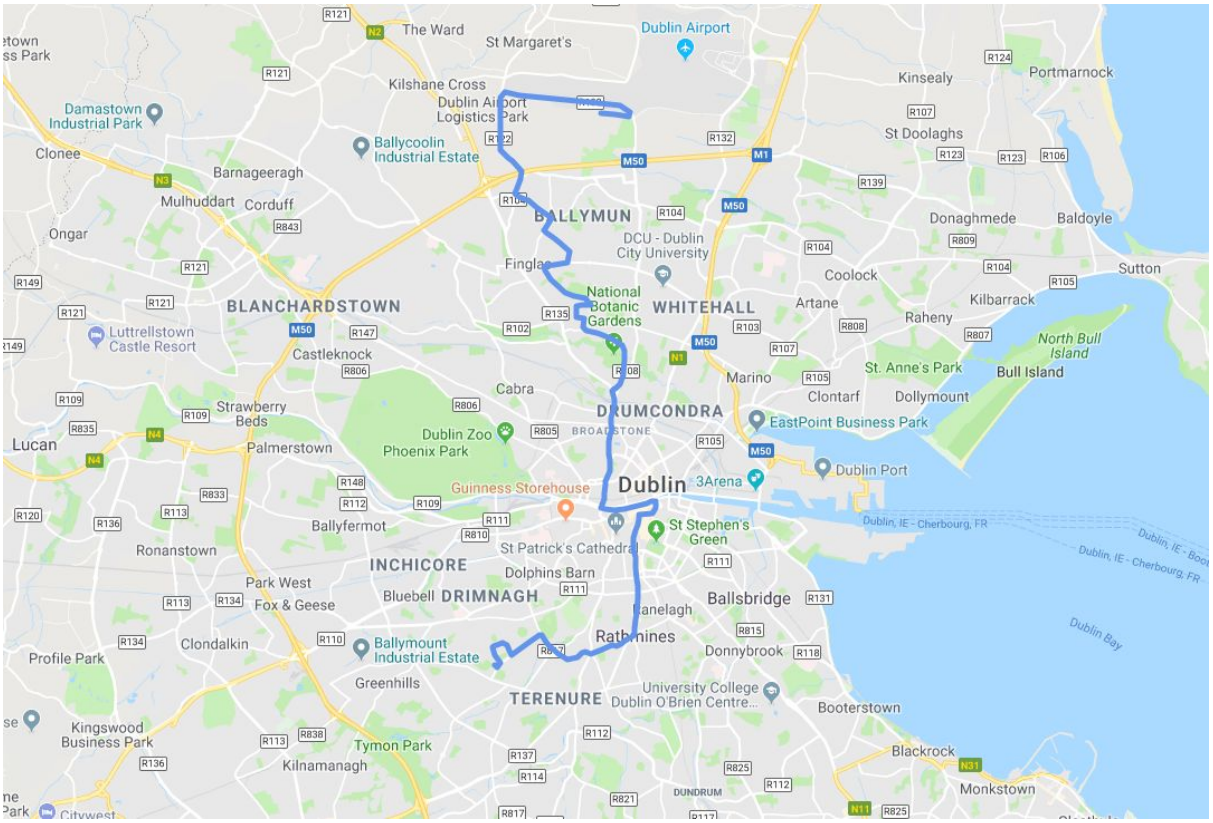

## 00151001.html



## 00160001.html




## 083A1001.html



# Ερώτημα 2

## (A-1) Εύρεση κοντινότερων γειτόνων

Σε αυτό το ερώτημα, στο αρχείο Question_2/DTW_NN.py βρίσκουμε τους 5 κοντινότερους γείτονες, για κάθε διαδρομή που υπάρχει στο αρχείο test_set_a1.csv, από το αρχείο train_set.csv και τους αναπαριστούμε σε ένα αρχείο html. Αρχικά απεικονίζουμε την διαδρομή για την οποία ψάχνουμε τους 5 κοντινότερους γείτονες σε ένα αρχείο html (με την χρήση της βιβλιοθήκης gmplot), και στην συνέχεια ψάχνουμε και αναπαριστούμε τους 5 κοντινότερους γείτονες σε html αρχεία.

Για να τους βρούμε χρησιμοποιούμε την τεχνική Dynamic Time Warping (DTW), και για τον υπολογισμό της απόστασης χρησιμοποιούμε τον τύπο του Harvesine, με κάθε διαδρομή του αρχείου train_set.csv. Για την DTW χρησιμοποιήθηκε η βιβλιοθήκη που βρίσκεται στον εξής σύνδεσμο: https://github.com/pierre-rouanet/dtw και ο τύπος του Harversine έχει υλοποιηθεί στο αρχείο HarversineDistance.py, οπότε ο υπολογισμός των αποστάσεων γίνεται με την εξής συνάρτηση:

```
dtw(query, journey, dist=lambda spot1, spot2: haversine
(spot1[1], spot1[2], spot2[1], spot2[2]))
```

όπου spot1, spot2 είναι σημεία της τροχιάς του query(: η διαδρομή για την οποία ψάχνουμε τους κοντινότερους γείτονες) και του journey(:η διαδρομή του train_set.csv την οποία εξετάζουμε) αντίστοιχα. Το spot1[1] περιέχει την lon τιμή και το spot1[2] περιέχει την lat τιμή του σημείου spot1, ομοίως και το spot2.

Στην συνέχεια αποθηκεύουμε τα αποτελέσματα της DTW σε μια λίστα μαζί με έναν δείκτη, με τον οποίο θα μπορούμε να βρούμε την διαδρομή με αυτήν την απόσταση από το αρχείο train_set.csv. Αφού έχουμε βρεί όλες τις αποστάσεις των διαδρομών του αρχείου train_set.csv από την διαδρομή που ψάχνουμε, ταξινομούμε την λίστα ώστε να είναι αύξουσα, και έπειτα διαλέγουμε τα 5 πρώτα και τα αναπαρηστούμε με την χρήση της gmplot.

Αφού έχουμε αναπαραστήσει τις διαδρομές των 5 κοντινότερων γειτόνων δημιουργούμε το html αρχείο Question_2/Q2A1_produced/finals/final_(query_id).html, και με την χρήση iframes απεικονίζουμε την διαδρομή μας και τις διαδρομές των κοντινότερων γειτόνων.

# Αποτελέσματα

## Question_2/Q2A1_produced/finals/final_1.html


Test Trip 1
Δt= 1171.5758771896362sec


Neighbor 1
JP_ID: 01501001
DTW: 0.0

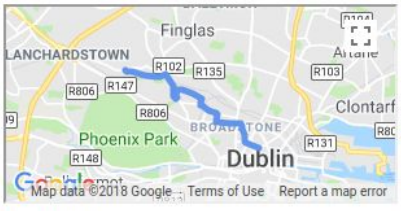

Neighbor 2
JP_ID: 01501001
DTW: 0.013874606415368753


Neighbor 3
JP_ID: 01501001
DTW: 0.015224048129470756

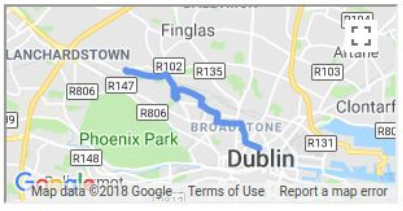

Neighbor 4
JP_ID: 01501001
DTW: 0.015736379836091052


Neighbor 5
JP_ID: 01501001
DTW: 0.016013091332028724

## Question_2/Q2A1_produced/finals/final_2.html



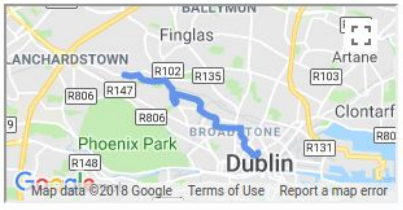

Test Trip 2
Δt= 608.85720038414sec




Neighbor 1
JP_ID: 01200001
DTW: 0.0


Neighbor 2
JP_ID: 01200001
DTW: 0.020685469224294074


Neighbor 3
JP_ID: 01200001
DTW: 0.022780762378723198




Neighbor 4
JP_ID: 01200001
DTW: 0.02432501949003578


Neighbor 5
JP_ID: 01200001
DTW: 0.024388564877253417

## Question_2/Q2A1_produced/finals/final_3.html


Test Trip 3
Δt= 1143.3212985992432sec


Neighbor 1
JP_ID: 00791001
DTW: 0.0


Neighbor 2
JP_ID: 00791001
DTW: 0.015336994044971993


Neighbor 3
JP_ID: 00791001
DTW: 0.01658076910137492


Neighbor 4
JP_ID: 00791001
DTW: 0.01710115456587192


Neighbor 5
JP_ID: 00791001
DTW: 0.017658909723781403

## Question_2/Q2A1_produced/finals/final_4.html


Test Trip 4
Δt= 760.397617816925sec


Neighbor 1
JP_ID: 00010002
DTW: 0.0


Neighbor 2
JP_ID: 00010002
DTW: 0.014882175877381278


Neighbor 3
JP_ID: 00010002
DTW: 0.017247889261746016


Neighbor 4
JP_ID: 00010002
DTW: 0.019696952751058337


Neighbor 5
JP_ID: 00010002
DTW: 0.019894708089114727

Question_2/Q2A1_produced/finals/final_5.html


Test Trip 5
Δt= 742.2270722389221sec

Neighbor 1
JP_ID: 01300001
DTW: 0.0

Neighbor 2
JP_ID: 01300001
DTW: 0.024293949698498574


Neighbor 3
JP_ID: 01300001
DTW: 0.02481811786358962

Neighbor 4
JP_ID: 01300001
DTW: 0.02561600605820375

Neighbor 5
JP_ID: 01300001
DTW: 0.02594746426365681

## (A-2) Εύρεση κοντινότερων υποδιαδρομών

Σε αυτό το ερώτημα, το οποίο βρίσκεται στο αρχείο Question_2/kcommonNeighbors.py, βρίσκουμε για κάθε μια από τις 5 διαδρομές που υπάρχουν στο αρχείο test_set_a2.csv, τις 5 διαφορετικές διαδρομές οι οποίες έχουν την μεγαλύτερη κοινή υποδιαδρομή από όλες τις διαδρομές που βρίσκονται στο αρχείο train_set.csv και τις απεικονίζουμε σε ένα αρχείο .html. Αρχικά, παίρνουμε μια μια κάθε διαδρομή από το αρχείο test_set_a2.csv και την αναπαριστούμε σε ένα αρχείο .html (με την χρήση της βιβλιοθήκης gmplot). Στην συνέχεια, για κάθε διαδρομή που ειναι μεσα στο αρχείο train_set.csv καλούμε την συνάρτηση LonsLatsLCS() η οποία δέχεται σαν όρισμα τις δύο διαδρομές (με την μορφή λίστας) και μας επιστρέφει το πλήθος των κοινών σημείων που έχουν οι δύο αυτές διαδρομές και μια λίστα η οποία περιέχει αυτά τα κοινά σημεία των διαδρομών [(time1, lons1, lats1), (time2, lons2, lats2) … ], τα οποία αποθηκεύονται σε μια λίστα ονόματι results[ ] μαζί με το index το οποίο προσδιορίζει την αντίστοιχη διαδρομή μέσα στο train_set. Από κάτω φαίνεται η δομή των περιεχομένων της λίστας results[ ] :

| Commons (int) |
| --- |
| CommonPath [ ] |
| index (int) |

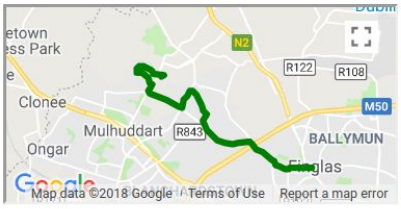
Η συνάρτηση LonsLatsLCS() λειτουργεί με την λογική του αλγορίθμου **longest common subsequence** (**LCS**) ωστόσο για να κάνει match δύο σημεία από τις διαφορετικές διαδρομές ελέγχει την απόσταση τους με την συνάρτηση του Harversine() η οποία επιστρέφει km, όπως αναλύθηκε και προηγουμένος, και αν αυτη δεν ξεπερναει τα 0.2km τοτε τα κάνει match. Έπειτα αφού γεμίσει τον πίνακα C με τις σωστές τιμές, βρίσκει την μεγαλύτερη τιμή η οποία βρίσκεται στην τέρμα κάτω δεξιά θέση, και καλεί την συνάρτηση FindCommonPath() η οποία παίρνει σαν όρισμα τον πίνακα C και μια από τις δύο διαδρομές και επιστρέφει μια λίστα με τη κοινή διαδρομή σε (time, lons, lats) μέσα από τον πίνακα C όπως λειτουργεί ο **LCS**. Τέλος η συνάρτηση LonsLatsLCS() επιστρέφει αυτές τις πληροφορίες.

Έπειτα, από αυτήν την λίστα (results[ ]) βρίσκουμε τις 5 διαδρομές με τα περισσότερα κοινά σημεία και τα αναπαριστούμε στον χάρτη όπως προηγουμένος. Για την αναπαράσταση αυτήν πάνω στον χάρτη γίνεται η χρήση των spot[0], spot[1] με των αντίστοιχων lons και lats όπως περιγράφηκε και προηγουμένος.

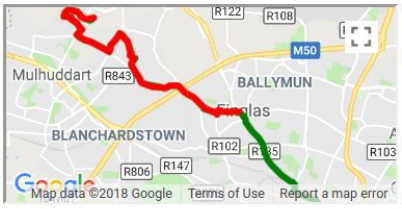
Αφού έχουμε αναπαραστήσει τις 5 επιθυμητές διαδρομές δημιουργούμε το html αρχείο Question_2/Q2A2_produced/finals/final (νούμερο του test).html, και με την χρήση iframes απεικονίζουμε την διαδρομή μας και τις διαδρομές των κοντινότερων γειτόνων.

## Αποτελέσματα

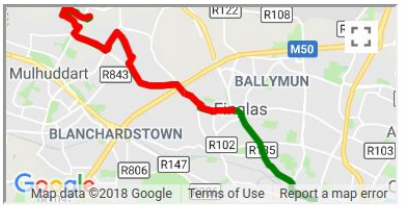
Question_2/Q2A2_produced/finals/final1.html



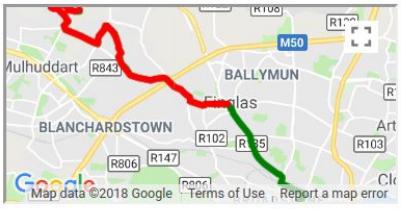

Test Trip 1
Dt= 231.447695017sec



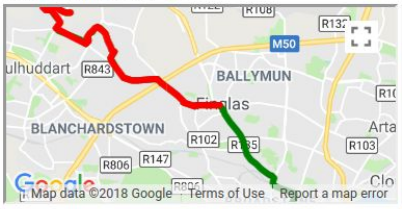

Neighbor 1
JP_ID: 040D1002
#Matching Points: 78



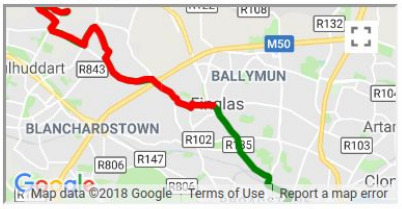

Neighbor 2
JP_ID: 040D1002
#Matching Points: 78




Neighbor 3
JP_ID: 040D1002
#Matching Points:76



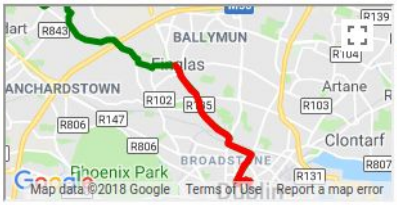

Neighbor 4
JP_ID: 040D1002
#Matching Points:76



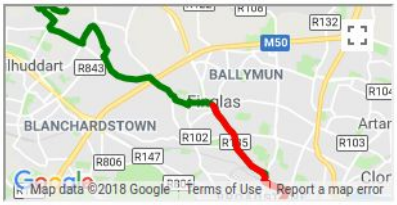

Neighbor 5
JP_ID: 040D1002
#Matching Points:75

## Question_2/Q2A2_produced/finals/final2.html

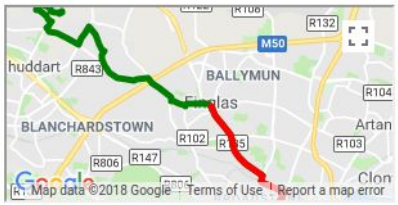

Test Trip 2
Dt= 248.3577981sec



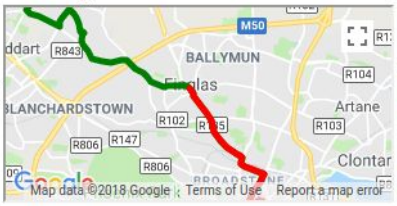

Neighbor 1
JP_ID: 040D1002
#Matching Points: 82



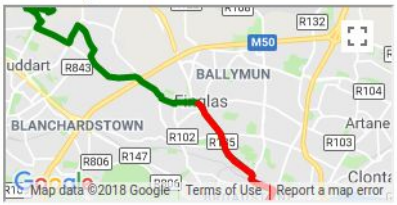

Neighbor 2
JP_ID: 040D1002
#Matching Points: 78




Neighbor 3
JP_ID: 040D1002
#Matching Points:75




Neighbor 4
JP_ID: 040D1002
#Matching Points:74

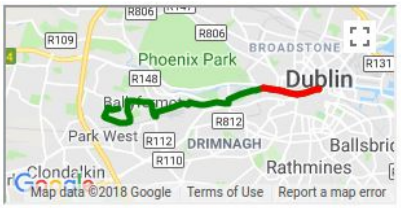



Neighbor 5
JP_ID: 040D1002
#Matching Points:73

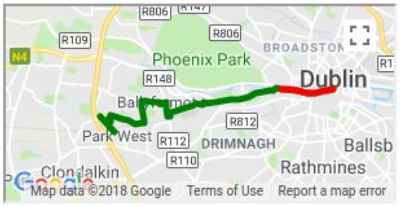
## Question_2/Q2A2_produced/finals/final3.html


Test Trip 3
Dt= 124.239289999sec


Neighbor 1
JP_ID: 01451008
#Matching Points: 40

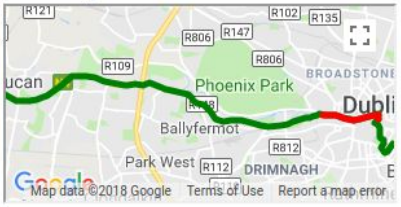



Neighbor 2
JP_ID: 00790001
#Matching Points: 40




Neighbor 3
JP_ID: 079A0001
#Matching Points:40


Neighbor 4
JP_ID: 01451001
#Matching Points:40




Neighbor 5
JP_ID: 067X0001
#Matching Points:40

## Question_2/Q2A2_produced/finals/final4.html


Test Trip 4
Dt= 174.066302061sec


Neighbor 1
JP_ID: 00790001
#Matching Points: 59

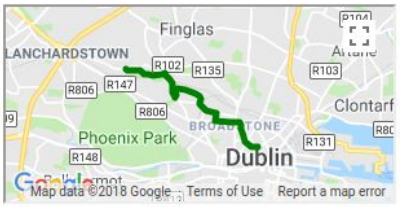

Neighbor 2
JP_ID: 00790001
#Matching Points: 59


Neighbor 3
JP_ID: 00790001
#Matching Points:59


Neighbor 4
JP_ID: 00790001
#Matching Points:59


Neighbor 5
JP_ID: 00790001
#Matching Points:59

## Question_2/Q2A2_produced/finals/final5.html




Test Trip 5
Dt= 210.066271067sec


Neighbor 1
JP_ID: 01200001
#Matching Points: 73


Neighbor 2
JP_ID: 01200001
#Matching Points: 73


Neighbor 3
JP_ID: 01200002
#Matching Points:72


Neighbor 4
JP_ID: 01200001
#Matching Points:71


Neighbor 5
JP_ID: 01200001
#Matching Points:71

# Ερώτημα 3

## Κατηγοριοποίηση

Για αυτό το ερώτημα έχουμε δημιουργήσει τρία αρχεία, τα Question_3/Cross_Validation.py, Question_3/Classification.py, Question_3/KNN.py.

Σχετικά με τον αλγόριθμο του KNN, αρχικά δοκιμάσαμε να χρησιμοποιήσουμε τον αλγόριθμο του sklearn και προσπαθήσαμε να προσαρμόσουμε τα δεδομένα ώστε να είναι αποδεκτά από τον KNN του sklearn. Η ιδέα ήταν ότι θα κρατήσουμε τον πίνακα X που περιέχει τα δεδομένα μας ως global και στον KNN θα στείλουμε έναν πίνακα με δείκτες σε αυτόν. Το περιεχόμενο του πίνακα θα είναι της εξής μορφής

[0, 0 ,1, 2, 3, 4, 5,...]
[1, 0 ,1, 2, 3, 4, 5,...]
[2, 0 ,1, 2, 3, 4, 5,...]
[3, 0 ,1, 2, 3, 4, 5,...]
.
.
.

Η πρώτη τιμή της κάθε λίστας δείχνει σε ποιά γραμμή του X κοιτάμε και οι υπόλοιπες είναι δείκτες για το σημείο της γραμμής που μελετάμε. Επίσης κάθε φορά θα πρέπει να ελέγχουμε αν ο δείκτης είναι μεγαλύτερος από το μέγεθος της γραμμής του X. Ωστόσο ενώ κατάφερα να τα περάσω στον KNN, για κάποιο λόγο άλλαζαν τα δεδομένα καθώς τα έστελνε στην συνάρτηση που υπολογίζει την απόσταση. Για αυτό τον λόγο δεν καταφέραμε να χρησιμοποιήσουμε τον KNN του sklearn, αντ αυτού δημιουργήσαμε την δικιά μας υλοποίηση του KNN στο αρχείο KNN.py. Για τον υπολογισμό της απόστασης χρησιμοποιεί την τεχνική DTW από την βιβλιοθήκη που χρησιμοποιήθηκε στον ερώτημα 2 Α-2, και τον τύπο του Harversine. Επίσης για voting scheme χρησιμοποιήθηκε majority voting, δηλαδή επιλέγεται το label που έχουν οι περισσότεροι από τους 5 κοντινότερους γειτόνους.

Το αρχείο Question_3/Cross_Validation.py εκτελεί 10-fold Cross validation με την χρήση της StratifiedKFold, επίσης γίνεται χρήση του δικού μας KNN (επίσης περιέχει σε σχόλια τον κώδικα με τον οποίο προσπαθήσαμε να εκτελέσουμε τον KNN του sklearn). Επειδή αργούσε υπερβολικά πολύ με είσοδο ολόκληρο το train_set, το δοκιμάσαμε με το 5% του ,όπως αναφέρθηκε στο piazza, και μας έδωσε Accuracy: 0.9249991539684694.

Το αρχείο Question_3/Classification.py ταξινομεί τις διαδρομές του test_set_a2.csv, δεδομένου του train_set.csv και παράγει το αρχείο testSet_JourneyPatternIDs.csv, το οποίο περιέχει:

| Test_Trip_ID | Predicted_JourneyPatternID |
|---|---|
| 1 | 040D1002 |
| 2 | 01201001 |
| 3 | 01511003 |
| 4 | 00790001 |
| 5 | 01200001 |