ΜΙΧΑΗΛ ΜΠΙΖΙΜΗΣ AM: **1115201500102**

ΚΩΝΣΤΑΝΤΙΝΤΟΣ ΤΣΙΑΡΑΣ AM: **1115201500165**

ΤΕΧΝΙΚΕΣ ΕΞΟΡΥΞΗΣ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΙΑ 2η

Classification report

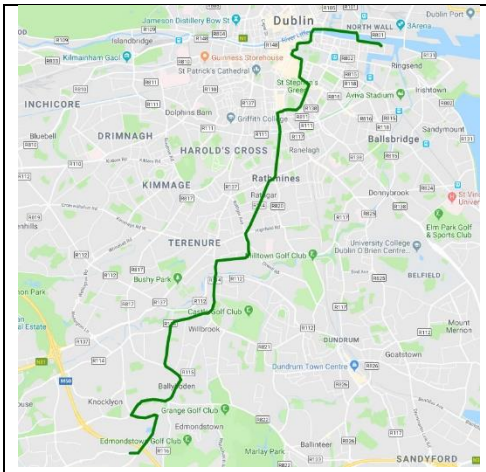
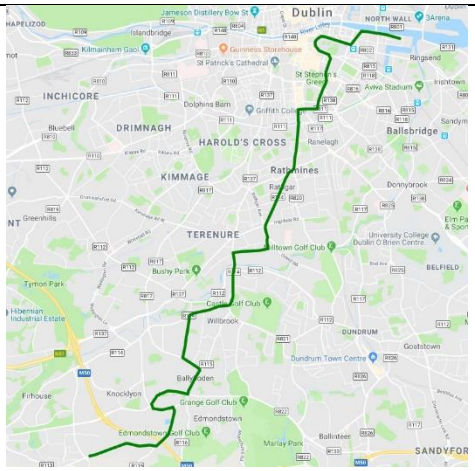
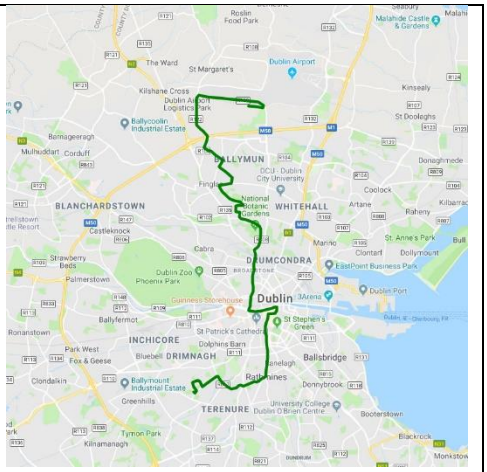
Κώδικας

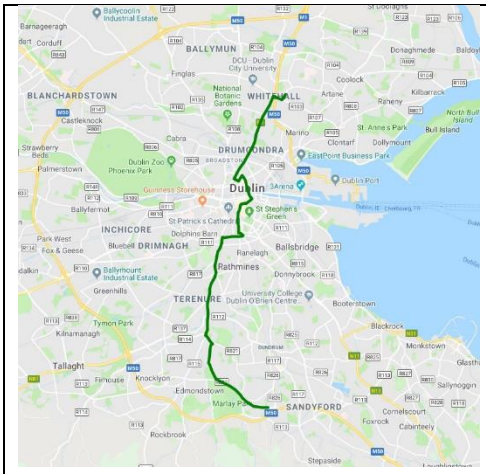
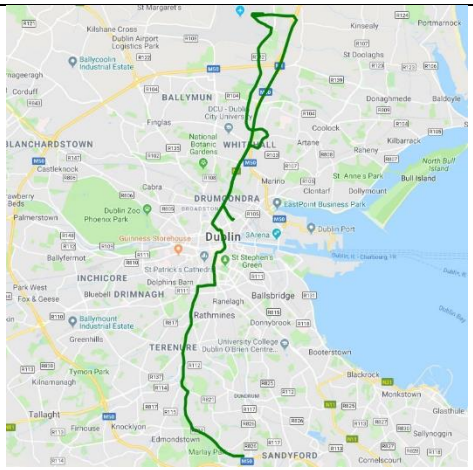
- Το αρχείο *visualization.py* το οποίο πραγματοποιεί την οπτικοποίηση των 5 πρώτων διαδρομών (με διαφορετικά journeyPatternId) του *train_set.csv* όπως ζητείται από το Ερώτημα 1.
- Το αρχείο *dtw_neighbours.py* με την υλοποίηση του Ερωτήματος 2 (Α-1).
- Το αρχείο *LCSS.py* με την δικιά μας υλοποίηση του αλγορίθμου LCSS και το αρχείο *lcss_neighbours.py* που κάνει χρήση αυτού για την υλοποίηση του Ερωτήματος 2 (Α-2).
- Το αρχείο *KNN.py* με την δικιά μας υλοποίηση του KNN.
- Τα αρχεία *knn_classification.py* και *knn_crossval.py* τα οποία παράγουν το ζητούμενο αρχείο "*testSet_JourneyPatternIDs.csv*" και πραγματοποιούν 10-fold Cross Validation αντίστοιχα, όπως περιγράφεται στο Ερώτημα 3.
- Το αρχείο *util.py* όπου υλοποιούνται συναρτήσεις που χρησιμοποιούνται από πολλαπλά αρχεία όπως η *harvesineDist()* και η *plotMap()*.

Για όλα τα ερωτήματα, τα datasets θεωρούμε πως υπάρχουν σε έναν φάκελο *./datasets/* στον φάκελο του παραδοτέου.

Ερώτημα 1

Οπτικοποίηση των Δεδομένων

		
JP_ID: 015B0001	JP_ID: 015B1001	JP_ID: 083A1001

 JP_ID: 00160001	 JP_ID: 00161001	

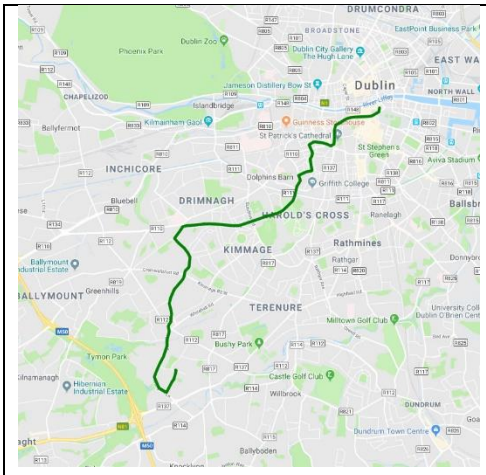
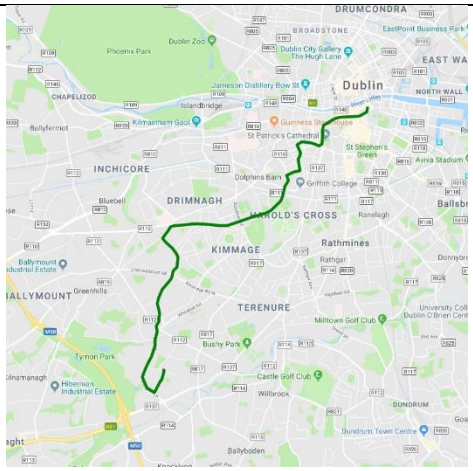
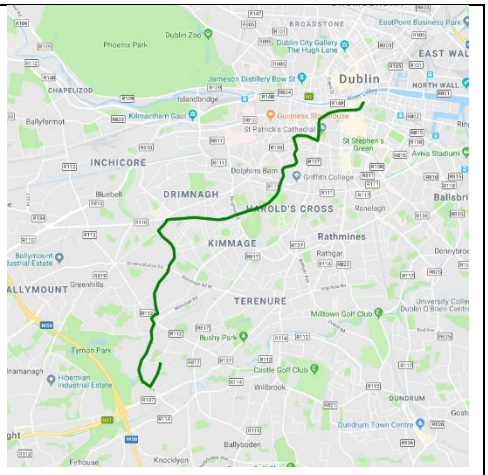
Ερώτημα 2 (A-1)

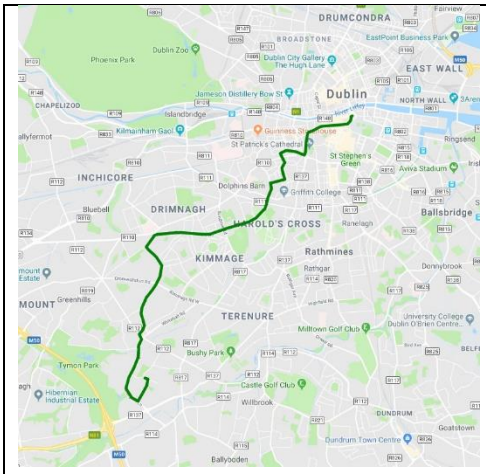
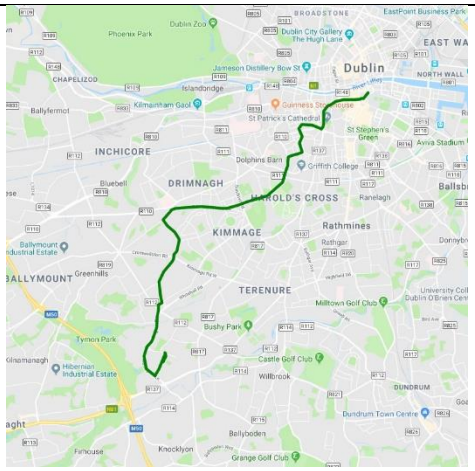
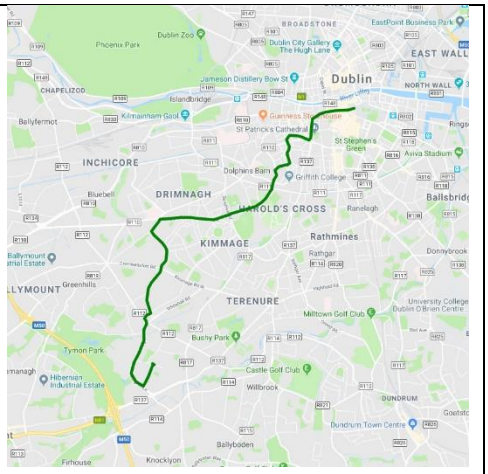
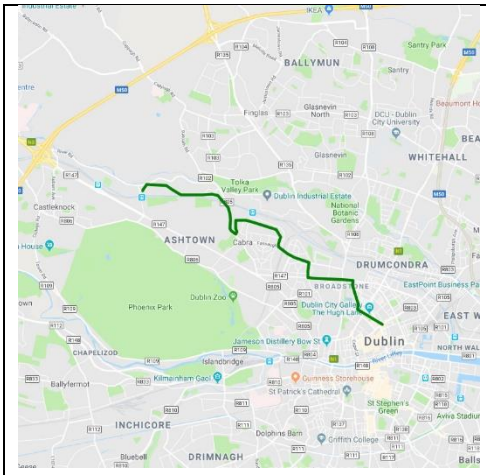
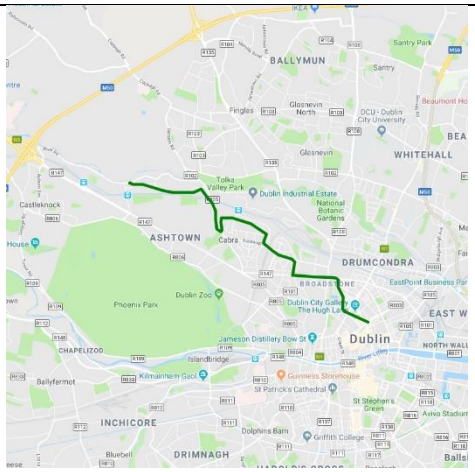
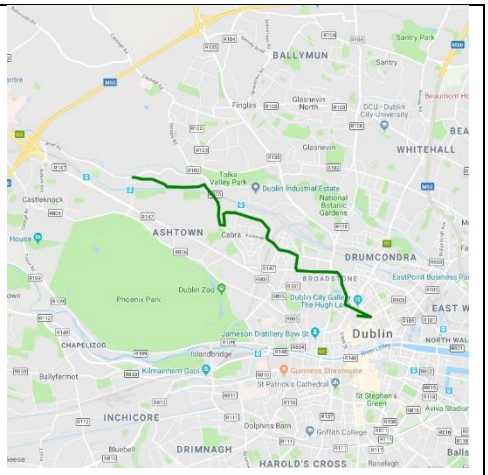
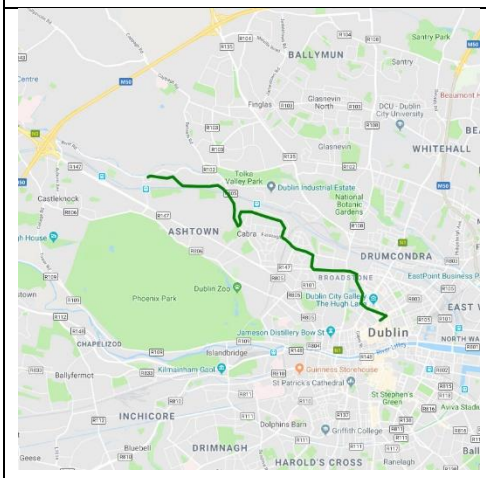
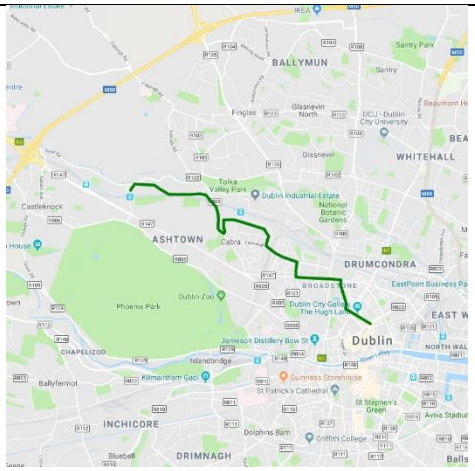
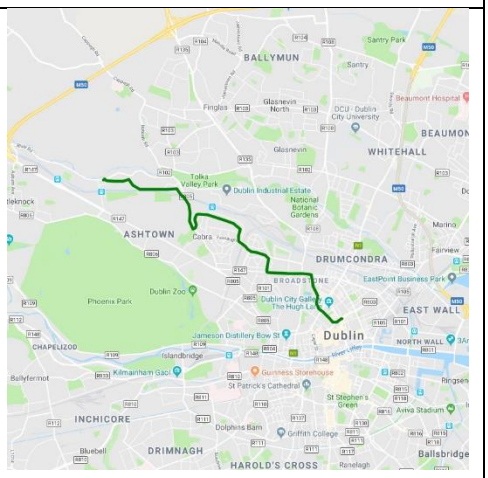
Εύρεση κοντινότερων γειτόνων

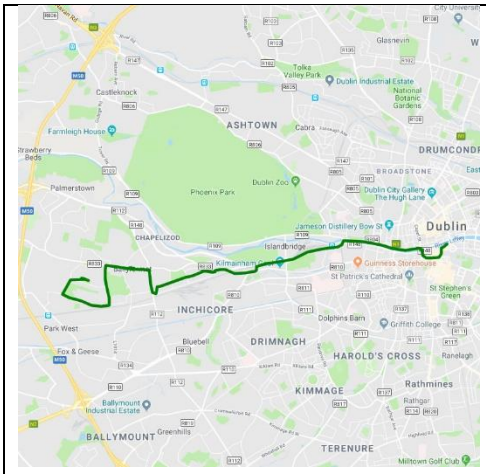
Λόγω ταχύτητας και επειδή χρησιμοποιείται και από το Ερώτημα 3, επιλέξαμε να χρησιμοποιήσουμε την έτοιμη βιβλιοθήκη fastdtw (https://pypi.org/project/fastdtw/).

Ο DTW κάνει «stretch» δυο χρονοσειρές (εδώ: διαδρομές) κατάλληλα και επιστρέφει την απόκλισή τους. Όσο μικρότερη είναι η απόκλιση αυτή τόσο πιο παρόμοιες είναι δύο χρονοσειρές.

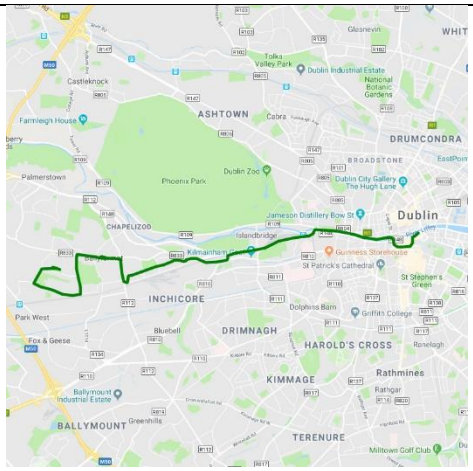
Παρατηρούμε για όλα τα Test Trips του test_set οι 5 κοντινότεροι γείτονες βάσει DTW προκύπτουν να έχουν αρκετά παρόμοιες διαδρομές και μάλιστα το ίδιο JPID μεταξύ τους, πράγμα λογικό αφού τέτοιες διαδρομές μπορούν να γίνουν «stretched» ώστε να έχουν μικρή έως ελάχιστη απόκλιση.

 Test Trip 1 Δt = 67 sec	 *Neighbor 1* JP_ID: 01501001 DTW: 0.0km	 *Neighbor 2* JP_ID: 01501001 DTW: 3.5km

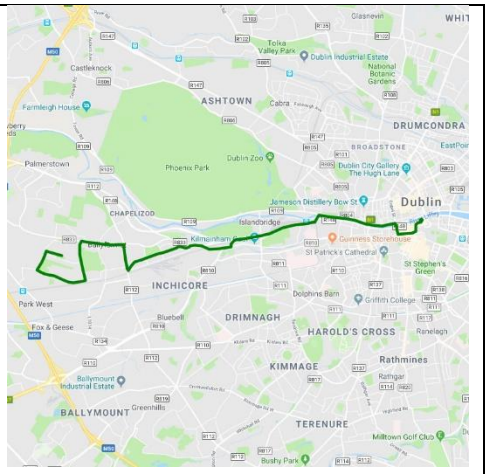
 *Neighbor 3* JP_ID: 01501001 DTW: 3.8km	 *Neighbor 4* JP_ID: 01501001 DTW: 4km	 *Neighbor 5* JP_ID: 01501001 DTW: 4.1km
 Test Trip 2 Δt = 41 sec	 *Neighbor 1* JP_ID: 01200001 DTW: 0.0km	 *Neighbor 2* JP_ID: 01200001 DTW: 2.8km
 *Neighbor 3* JP_ID: 01200001 DTW: 3.4km	 *Neighbor 4* JP_ID: 01200001 DTW: 3.4km	 *Neighbor 5* JP_ID: 01200001 DTW: 3.5km



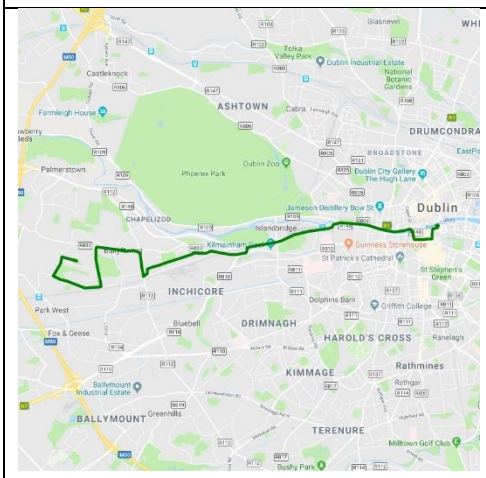
Test Trip 3
Δt = 51 sec



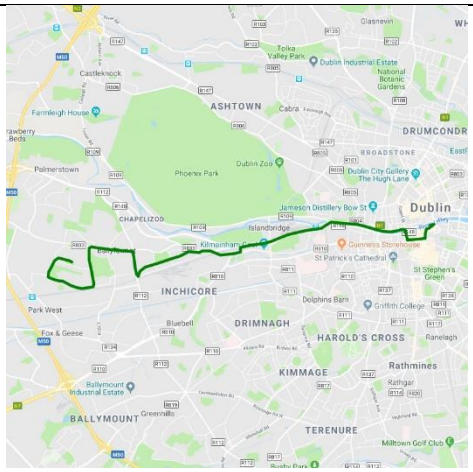
Neighbor 1
JP_ID: 00791001
DTW: 0.0km



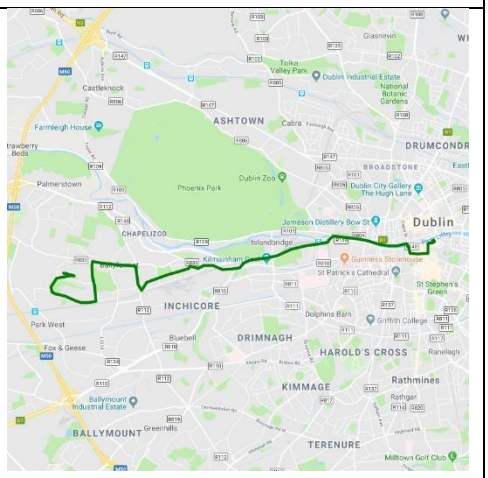
Neighbor 2
JP_ID: 00791001
DTW: 4.7km



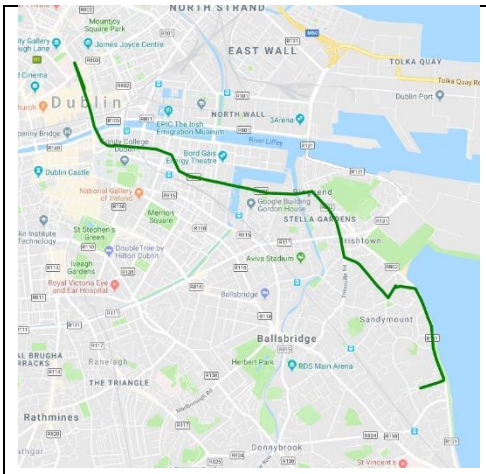
Neighbor 3
JP_ID: 00791001
DTW: 4.8km



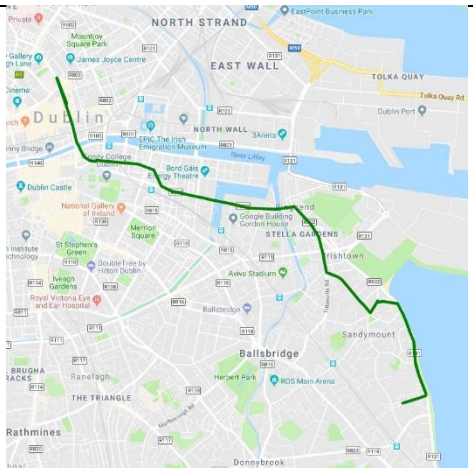
Neighbor 4
JP_ID: 00791001
DTW: 4.8km



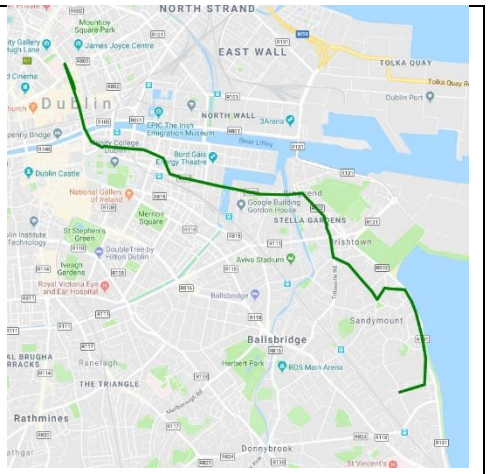
Neighbor 5
JP_ID: 00791001
DTW: 4.9km



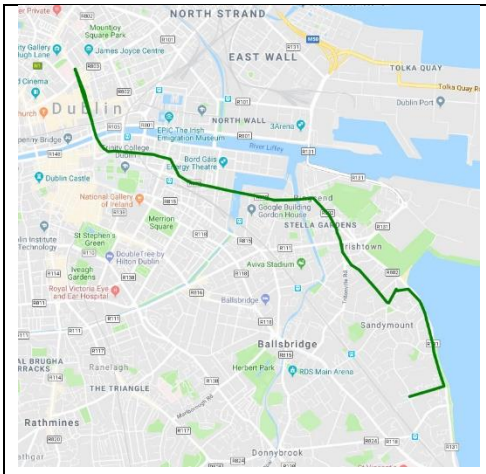
Test Trip 4
Δt = 42 sec



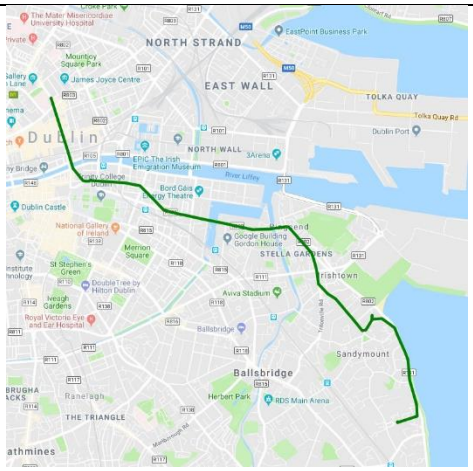
Neighbor 1
JP_ID: 00010002
DTW: 0.0km



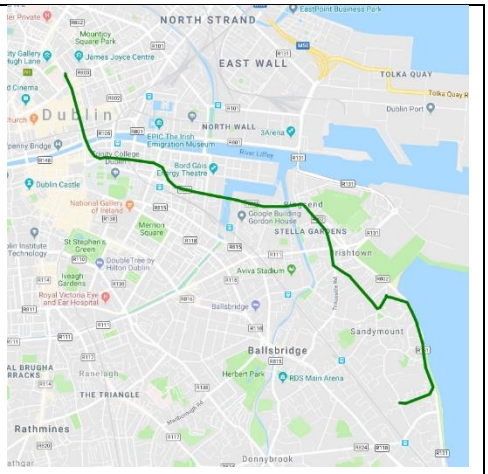
Neighbor 2
JP_ID: 00010002
DTW: 2.5km



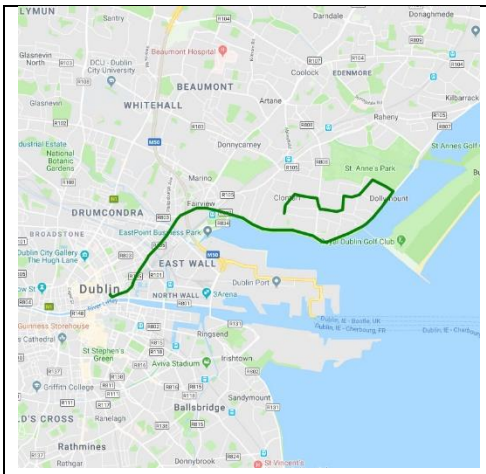
Neighbor 3
JP_ID: 00010002
DTW: 2.8km



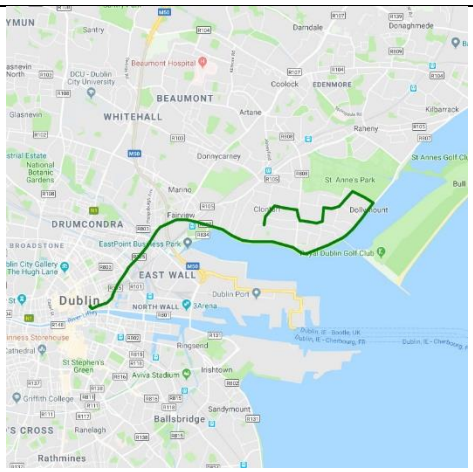
Neighbor 4
JP_ID: 00010002
DTW: 3.2km



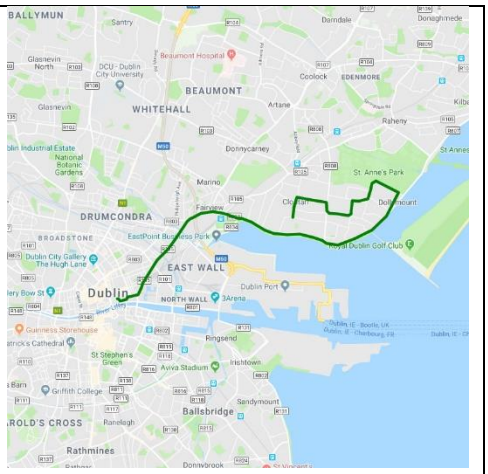
Neighbor 5
JP_ID: 00010002
DTW: 3.5km



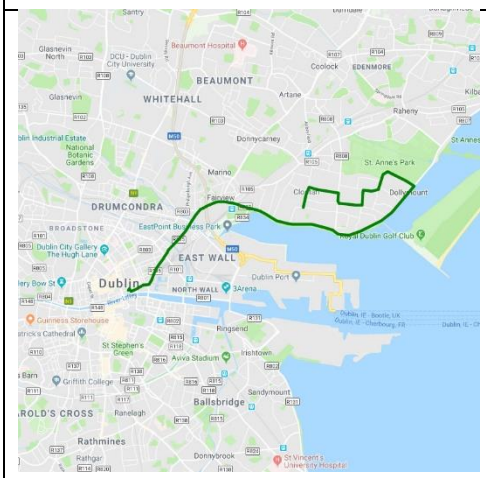
Test Trip 5
Δt = 40 sec



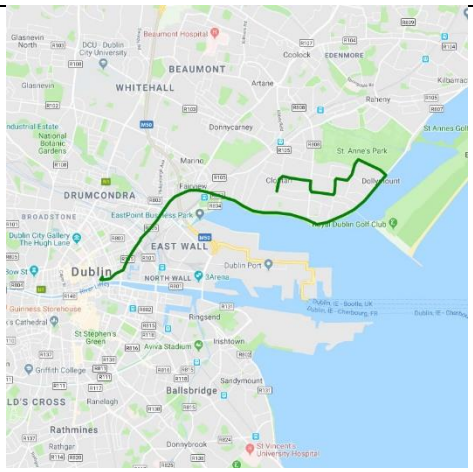
Neighbor 1
JP_ID: 01300001
DTW: 0.0km



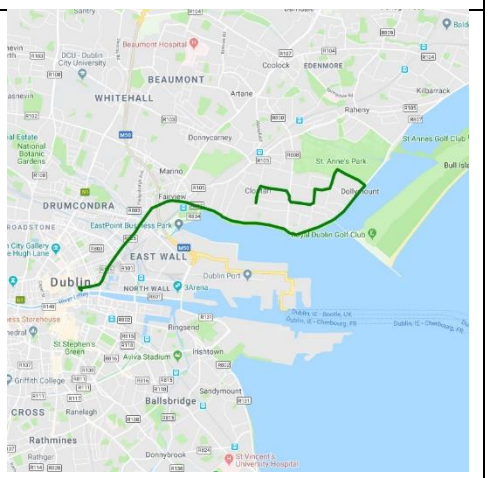
Neighbor 2
JP_ID: 01300001
DTW: 4.3km



Neighbor 3
JP_ID: 01300001
DTW: 4.5km



Neighbor 4
JP_ID: 01300001
DTW: 4.6km



Neighbor 5
JP_ID: 01300001
DTW: 4.7km

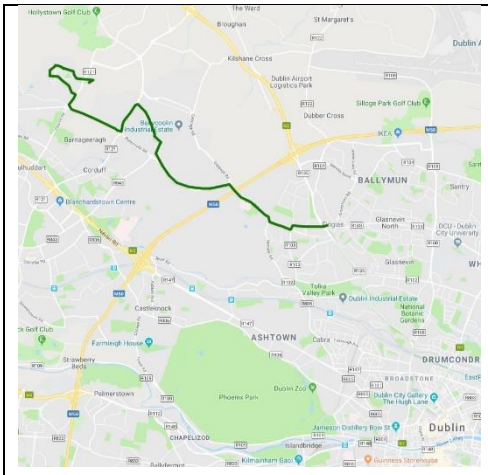
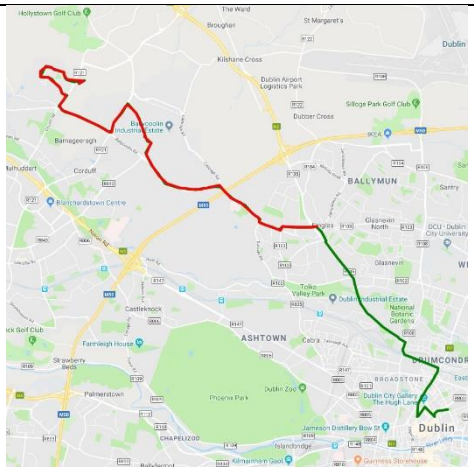
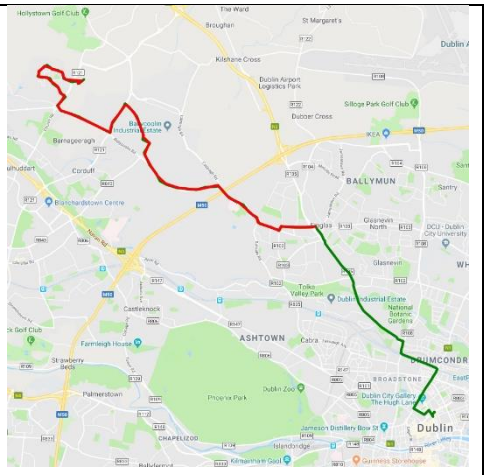
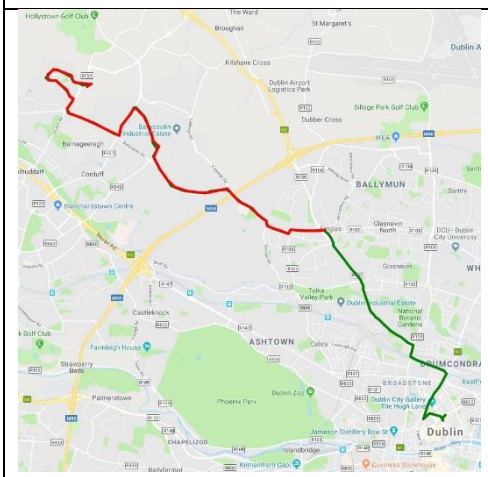
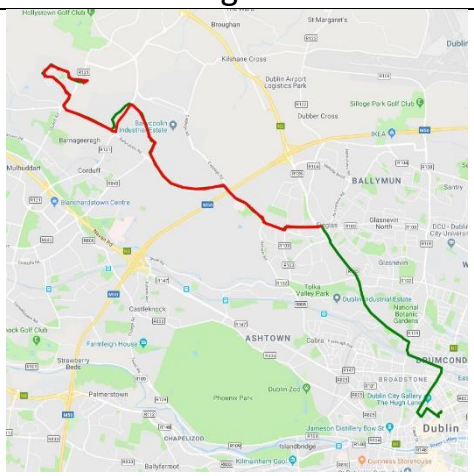
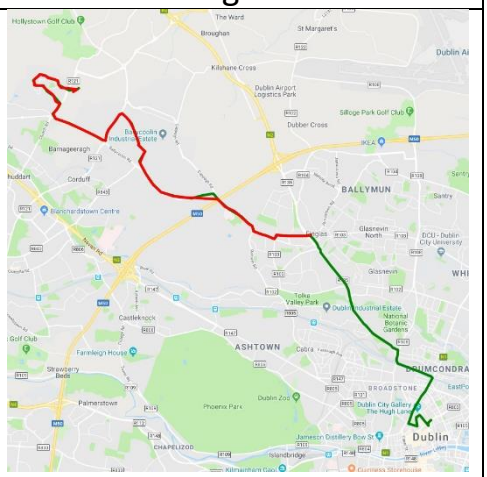
Ερώτημα 2 (A-2)

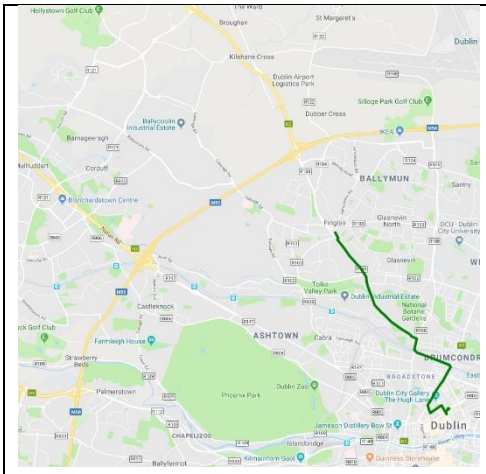
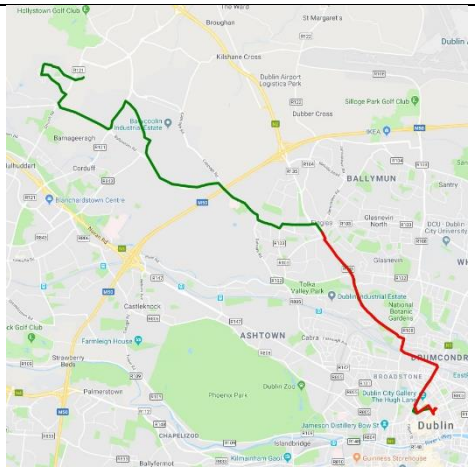
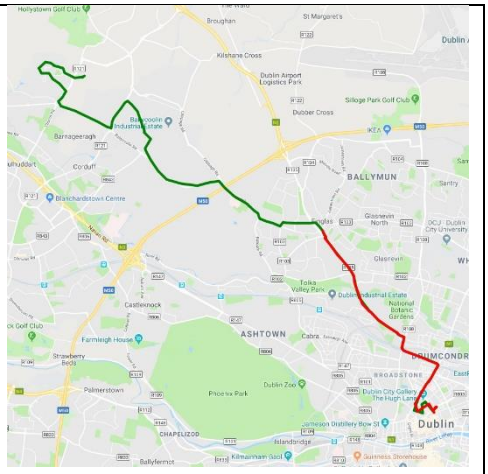
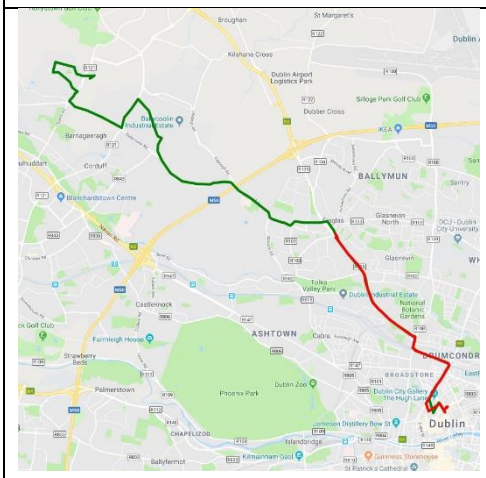
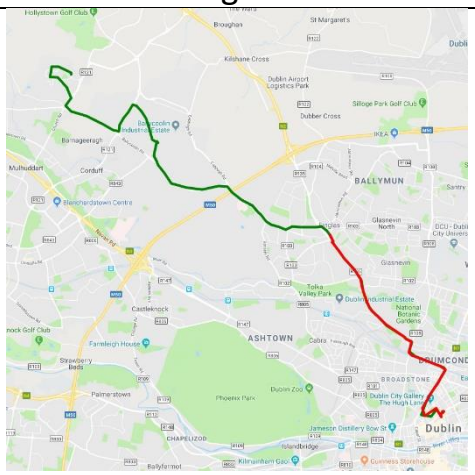
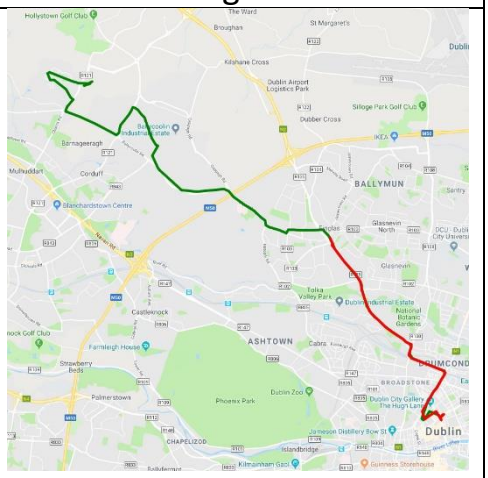
Εύρεση κοντινότερων υποδιαδρομών

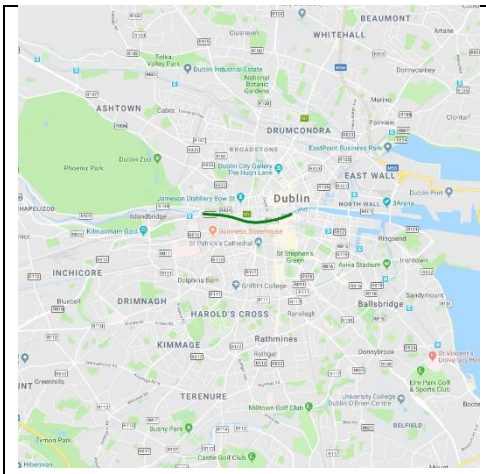
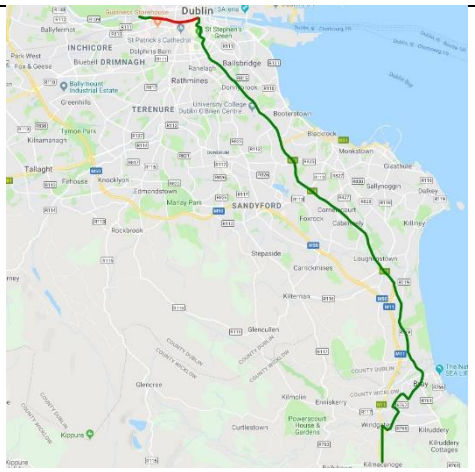
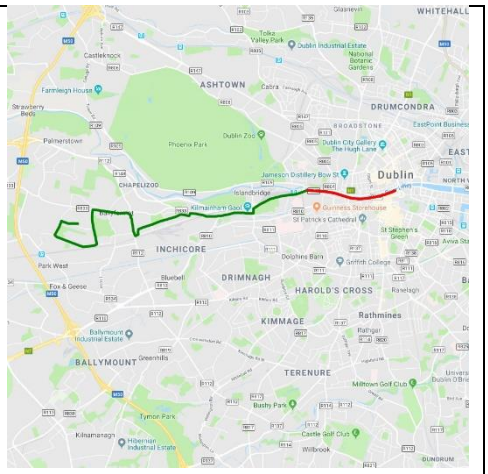
Τον LCSS τον υλοποιήσαμε εμείς στο *LCSS.py* .

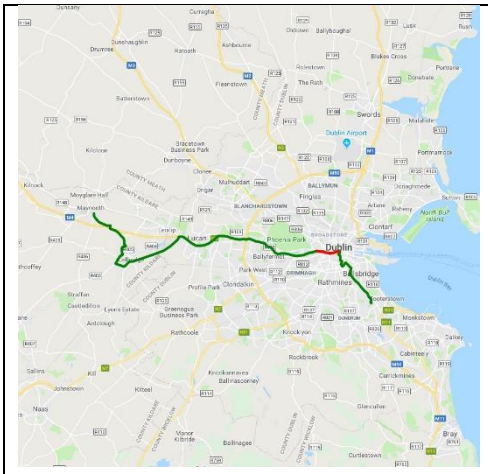
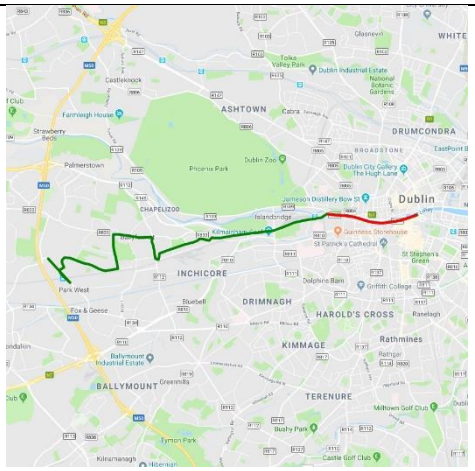
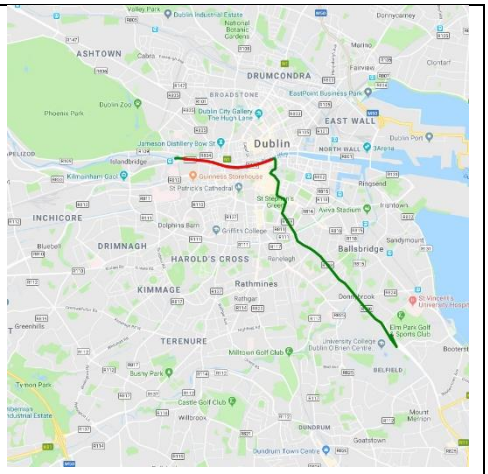
Ο LCSS επιστρέφει την μέγιστη κοινή υπακολουθία δύο χρονοσειρών (εδώ διαδρομών). Σημειώνεται ότι, στα πλαίσια της υλοποίησής μας, δύο σημεία στο χάρτη θεωρούνται «κοινά» εάν η Harversine απόσταση μεταξύ τους είναι μικρότερη ή ίση των 200m.

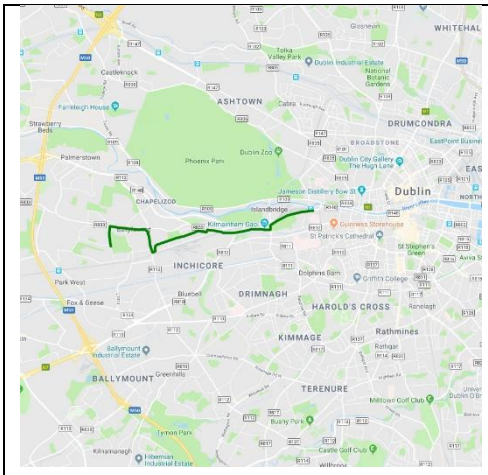
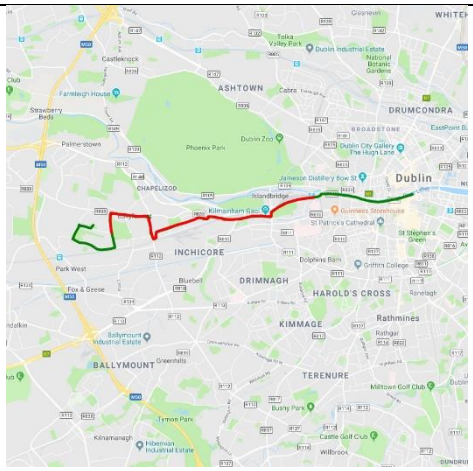
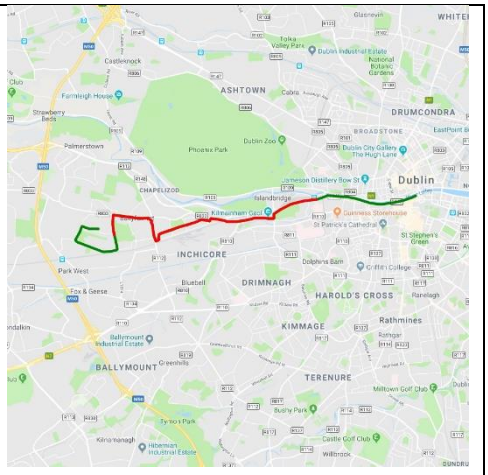
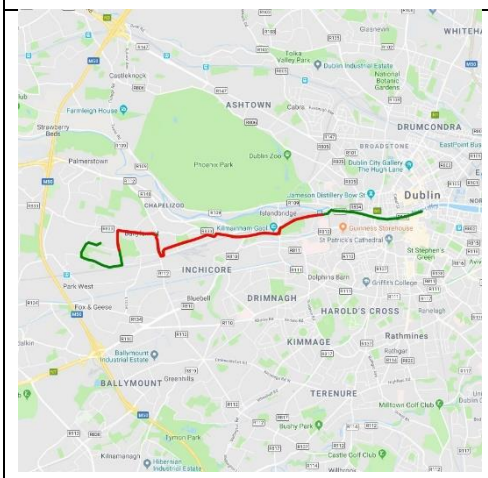
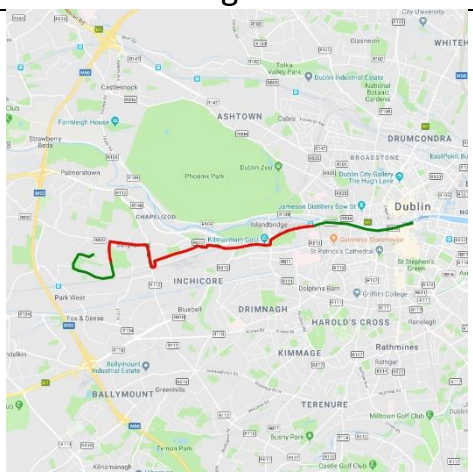
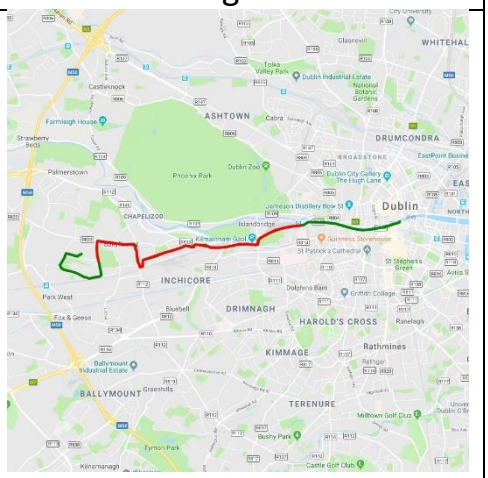
Εδώ παρατηρούμε ότι καθώς, σε αντίθεση με τον DTW, δεν ψάχνουμε ομοιότητα μεταξύ των διαδρομών, οι γείτονες που προκύπτουν για κάθε Test Trip του test_set σε κάποιες περιπτώσεις είναι αισθητά διαφορετικές διαδρομές μεταξύ τους, σε ορισμένες περιπτώσεις ακόμα και με διαφορετικά JPID. Αυτό είναι αναμενόμενο, αφού το μόνο που μας ενδιαφέρει εδώ είναι το πλήθος των matching points των δύο διαδρομών.

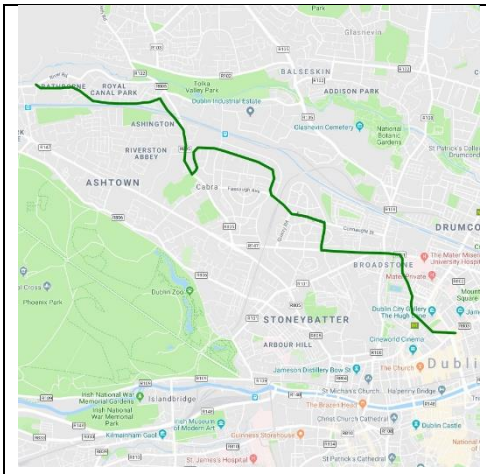
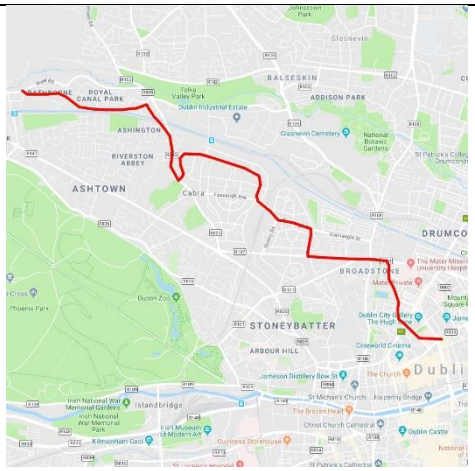
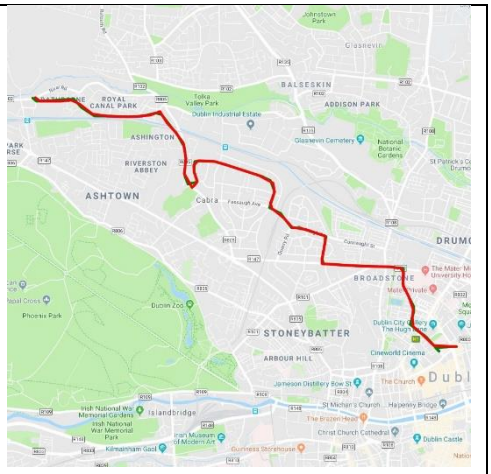
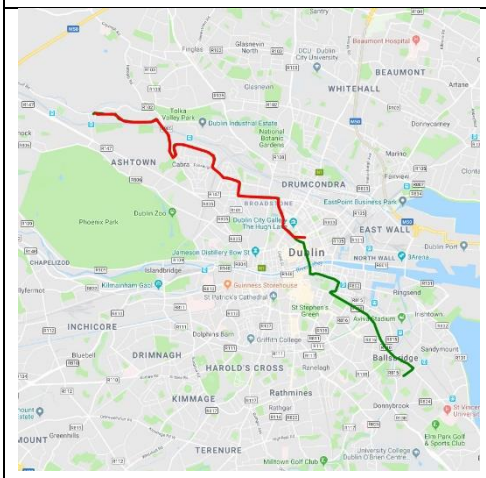
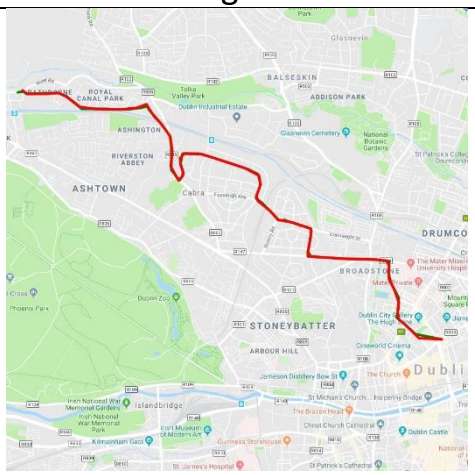
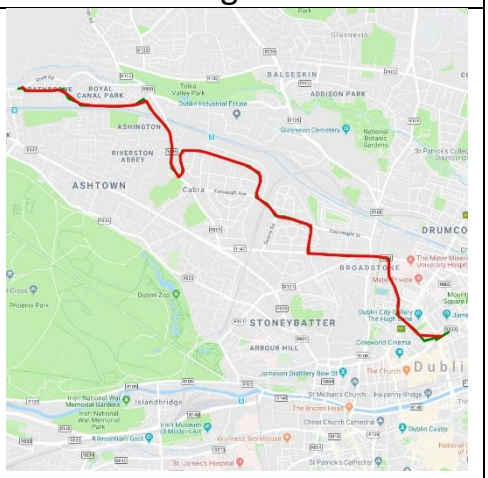
		
Test Trip 1 Δt = 348 sec	*Neighbor 1* JP_ID: 040D1002 #Matching Points: 78	*Neighbor 2* JP_ID: 040D1002 #Matching Points: 78
		
Neighbor 3 JP_ID: 040D1002 #Matching Points: 76	*Neighbor 4* JP_ID: 040D1002 #Matching Points: 76	*Neighbor 5* JP_ID: 040D1002 #Matching Points: 75

 Test Trip 2 Δt = 351 sec	 *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

 Test Trip 3 Δt = 180 sec	 *Neighbor 1* JP_ID: 01451001 #Matching Points: 40	 *Neighbor 2* JP_ID: 00790001 #Matching Points: 40

 *Neighbor 3* JP_ID: 067X0001 #Matching Points: 40	 *Neighbor 4* JP_ID: 079A0001 #Matching Points: 40	 *Neighbor 5* JP_ID: 01451008 #Matching Points: 40

 Test Trip 4 Δt = 265 sec	 *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

 Test Trip 5 Δt = 320 sec	 *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: 72	 *Neighbor 5* JP_ID: 01200001 #Matching Points: 71

Ερώτημα 3

Κατηγοριοποίηση

Για το Ερώτημα αυτό τροποποιήσαμε την δική μας υλοποίησή του KNN από την πρώτη εργασία με τις εξής διαφορές:

- Αντί για σημεία στο χώρο το *fit()* εφαρμόζεται πάνω σε χρονοσειρές (δηλαδή αποθηκεύουμε χρονοσειρές και όχι σημεία στο KNN clf).
- Αντί για EuclideanDistance μεταξύ των σημείων γίνεται χρήση της απόστασης που επιστρέφει η fastdtw μεταξύ χρονοσειρών με παράμετρο τη HarversineDistance.
- Αντί για heap sorting για εύρεση των top-K αποστάσεων κρατάμε ανά πάσα στιγμή τις K μικρότερες αποστάσεις σε έναν πίνακα μεγέθους K (το οποίο έχει καλύτερη πολυπλοκότητα – *O(Kn)* έναντι *O((n+k)logn)*).

Σε ό,τι αφορά το *Cross Validation*, προκειμένου να μπορέσουμε να εφαρμόσουμε Stratified K-fold (και συγκεκριμένα με *n_splits=10*) ήταν απαραίτητο να παραλείψουμε από το train_set τις διαδρομές εκείνες με journeyPatternId που εμφανίζονταν λιγότερες από 10 φορές. Αυτό δεν είναι παράλογο καθώς δεν θέλουμε να υπάρχει διαδρομή σε test fold με journeyPatternId που δεν εμφανίζεται καθόλου (ή εμφανίζεται ελάχιστες φορές) στα (9) train folds διότι έτσι δεν θα ήταν εξ ορισμού εφικτή η ορθή πρόβλεψή της.

Θεωρούμε ότι το accuracy που λαμβάνουμε έτσι ανταποκρίνεται στην πραγματικότητα δεδομένου ότι τα Test Trips που θα κληθεί να προβλέψει το πρόγραμμά μας θα έχουν journeyPatternIds τα οποία εμφανίζονται επαρκή αριθμό φορών (στην cross validation αυτός είναι τουλάχιστον 9) στο train_set ώστε να έχει γίνει fit για αυτά ο KNN.

Σε αυτό το σημείο, καθώς ο i-οστός γείτονας θα είναι πιο παρόμοιος με την Test χρονοσειρά από τον i+1 και δεδομένου ότι η διαφορά των δύο γειτόνων αυτών μπορεί να είναι και σημαντικά μεγάλη (πχ αν πρόκειται για διαφορετικές διαδρομές του train set), κρίναμε πως το majority voting που είχαμε χρησιμοποιήσει για *voting scheme* στην προηγούμενη εργασία για τον KNN ενδέχεται να μην είναι η καλύτερη επιλογή (ειδικά αν κληθούμε να μαντέψουμε journeyPatternId για το οποίο έχουμε λιγότερες απο K=5 διαδρομές στο train set). Για το λόγο αυτό πειραματιστήκαμε με διαφορετικές βαρύτητες για την ψήφο του καθενός και συγκεκριμένα με τα παρακάτω voting schemes, εξετάζοντας κάθε φορά το accuracy της 10-fold Cross Validation:

Βάρος ψήφου i-οστού γείτονα	Accuracy
1 *(Majority voting)*	0.983559
1/(K+i)	0.983559
1/dtwDist(Test, i)	0.985411
(K-i)/K	0.985411

Τελικά επιλέξαμε το τελευταίο από αυτά, δηλαδή το βάρος της ψήφου του i-οστού γείτονα να είναι ίσο με (K-i)/K , το οποίο δίνει μεν μεγαλύτερυ βαρύτητα στην ψήφο πχ του πρώτου γείτονα, αλλά επιτρέπει και «να μην περάσει το δικό του» αν οι περισσότεροι άλλοι γείτονες έχουν διαφορετικό journeyPatternId.

Τα αποτελέσματα της πρόβλεψής μας για το *test_set_a2.csv* (τα οποία επίσης περιλαμβάνονται στο αρχείο *testSet_JourneyPatternIDs.csv*) είναι τα παρακάτω:

Test_Trip_ID	Predicted_JourneyPatternID
0	02380002
1	01201001
2	01511003
3	00790001
4	01200001