

# READ ME

Αντώνης Σκαρλάτος sdi1400184

Για την παραγωγή των φωτογραφιών χρησιμοποιήθηκε το module 'selenium' και ο κώδικας βρίσκεται στο αρχείο convertHtmlToImage.py

## Ερώτημα 1

Ο κώδικας που παράγει τα .html αρχεία των πέντε διαφορετικών διαδρομών βρίσκεται στο αρχείο drawTrajectories.py



## Ερώτημα 2

Ο κώδικας για αυτό το ερώτημα βρίσκεται στον αρχείο search.py. Επίσης όλες οι αποστάσεις DTW, LCS, Haversine έχουν υλοποιηθεί από την αρχή και βρίσκονται στο αρχείο distances.py. Στην απόσταση

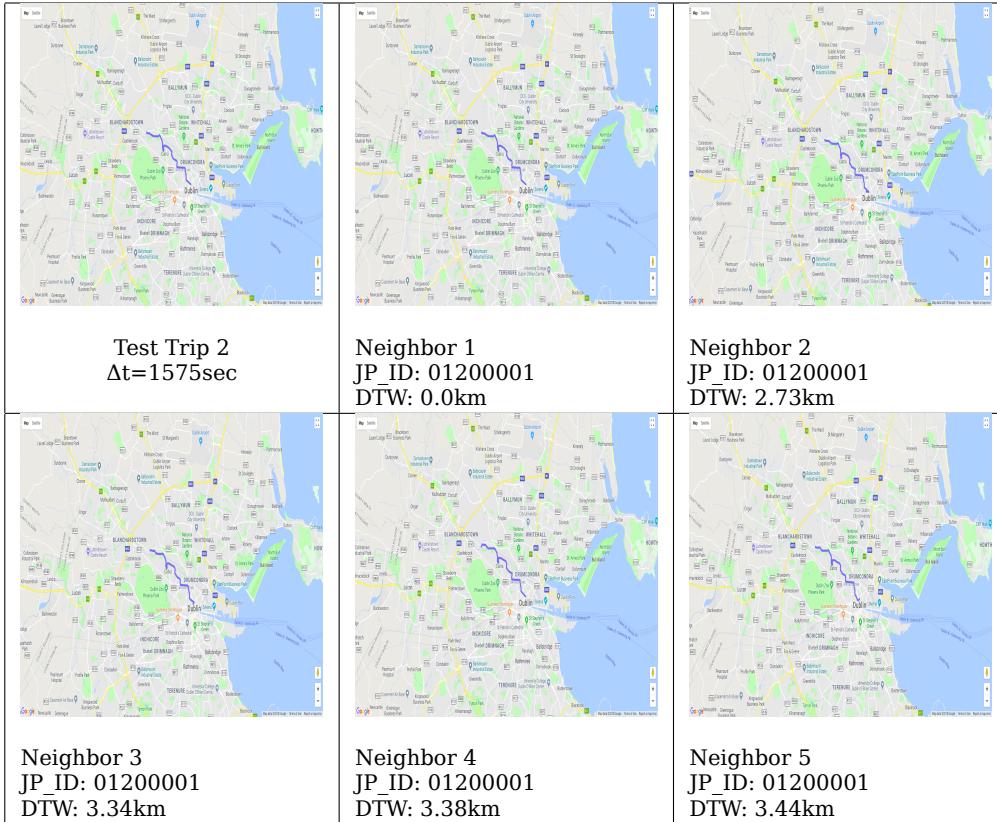
haversine γίνεται στρογγυλοποίηση στα δύο δεκαδικά ψηφία. Ο τρόπος που βρίσκονται οι γείτονες και για το DTW και για το LCS είναι μέσω bruteforce. Ελέγχονται όλοι οι γείτονες και ύστερα ταξινομούνται σε σχέση με την αποστασή τους. Στο τέλος διαλέγονται οι πέντε κοντινότεροι γείτονες και δημιουργούνται τα plots. Ο χρόνος  $\Delta t$  είναι κοινός για όλα τα plots γιατί μετράει πόσο χρόνο έκανε το πρόγραμμα για να βρει για όλο το testSet όλους τους κοντινότερους γείτονες.

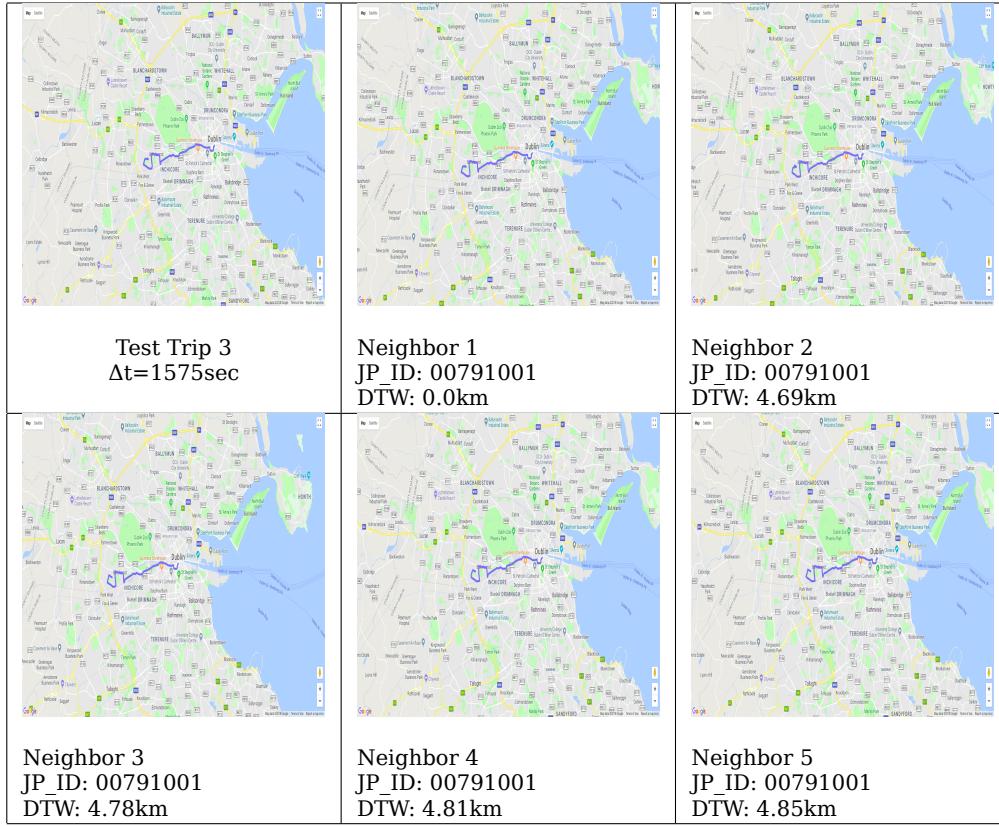
- Το DTW βρίσκει αρκετά ικανοποιητικά αποτελέσματα όπως φαίνεται και στις εικόνες παρακάτω. Ο λόγος είναι ότι βρίσκει το συνολικό κόστος που πρέπει να κάνει κάποια κομμάτια "stretch" για να μοιάζει η μία τροχιά στην άλλη.
- Το LCS βρίσκει το μέγιστο subsequence των δύο τροχιών. Όπως φαίνεται και στις εικόνες παρακάτω, σε αντίθεση με το DTW που θέλουμε να βρούμε το κόστος για να γίνουν οι τροχές παρόμοιες, στο LCS θέλουμε να δούμε κατά πόσο οι δύο τροχιές μοιάζουν σε κάποια κομμάτια και πόσα είναι τα σημεία αυτά.

Γενικά, φαίνεται ότι με το DTW παίρνουμε τον πλησιέστερο γείτονα, ενώ με το LCS παίρνουμε τις τροχιές που μοιάζουν σε κάποια κομμάτια, παρόλο που συνολικά μπορεί να είναι πολύ διαφορετικές στα υπόλοιπα κομμάτια.

## A-1)



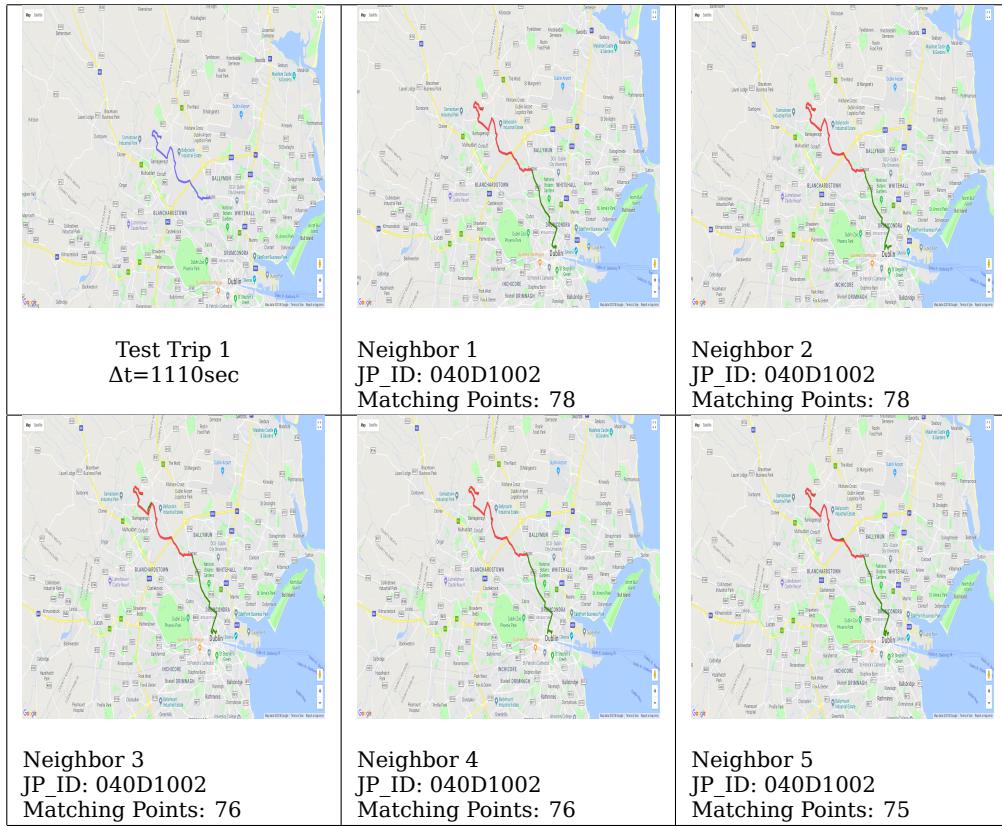


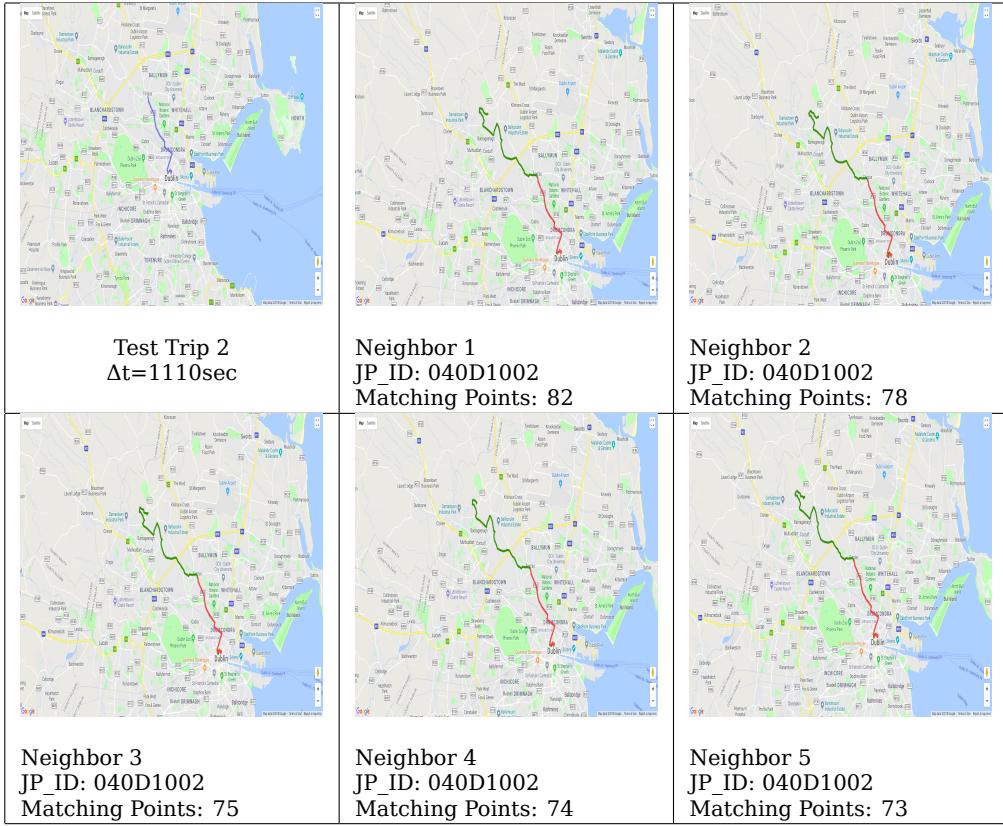






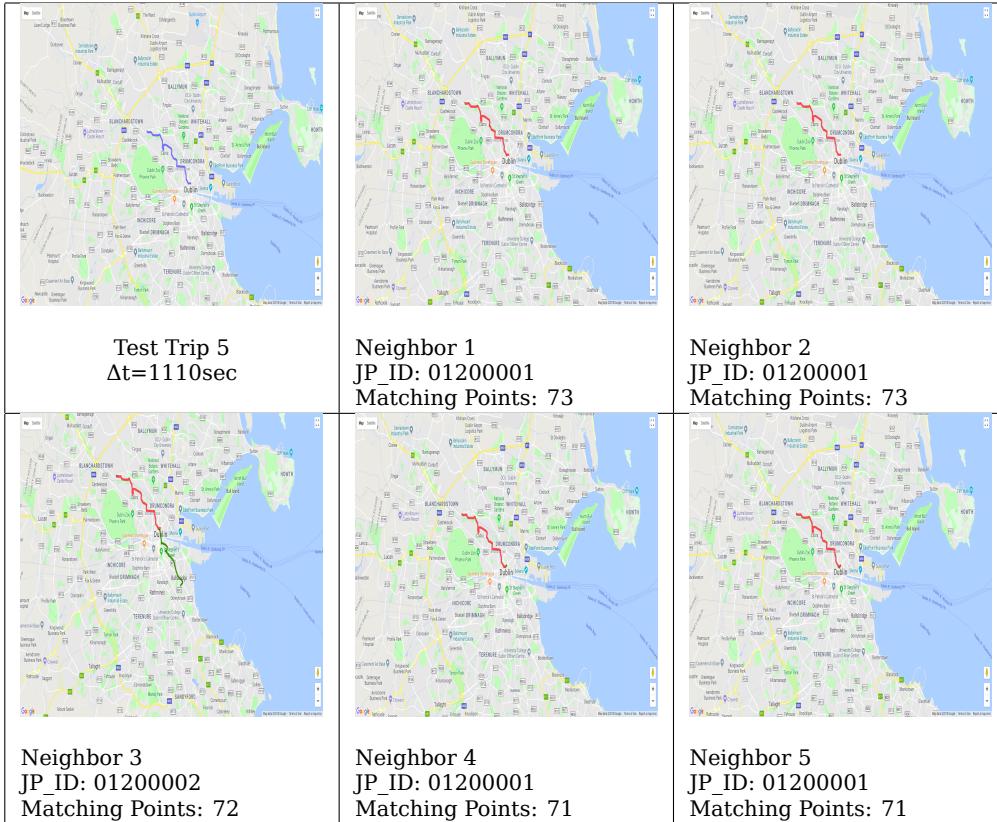
## A-2)











## Ερώτημα 3

Τα αποτελέσματα του classification βρίσκονται στο αρχείο testSet\_JourneyPatternIDs.csv To 10-cross-validation έγινε περίπου με το 5% του dataset.

### Voting schemes

- Ένα voting scheme που δοκιμάστηκε είναι να βλέπουμε ποιο patternId βρίσκεται τις περισσότερες φορές από τα πέντε κοντινότερα και να επιλέγουμε αυτό ως πλησιέστερο γείτονα. Σε ισοπαλίες διαλέγουμε τυχαία.

10-cross-validation accuracy = 80%

- Ένα άλλο voting scheme που δοκιμάστηκε είναι να βλέπουμε ποιο patternId έχει την μεγαλύτερη τιμή στο άθροισμα των  $1/(dist + 0.01)$  από τα πέντε κοντινότερα. Όσο πιο μικρό είναι το dist(δηλαδή πιο κοντά) τόσο μεγαλύτερο γίνεται το άθροισμα. Αυτό, σε αντίθεση με το προηγούμενο voting scheme βάζει κάποια βάρη. Αυτό σημαίνει ότι δεν μας ενδιαφέρει μόνο η σειρά που βρίσκονται οι γείτονες ως προς την απόσταση, αλλά και η ίδια η απόσταση ως τιμή, για την τελική επιλογή. Σε ισοπαλίες διαλέγουμε τυχαία.

10-cross-validation accuracy = 72%

Γενικά ο KNN είναι ένας αλγόριθμος που χρησιμοποιείται αρκετά στην πράξη επειδή είναι απλός και διατιθητικός, αλλά εξαρτάται για την απόδοση και η μετρική που χρησιμοποιείται. Στην περιπτωσή μας δεν παίρνουμε άσχημα αποτελέσματα αλλά ούτε και ιδανικά με βάση την απόδοση που μας δίνει το cross validation.