ΠΜΣ “Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης”

Καθηγητής: Τσιχριντζής Γεώργιος

Αναγνώριση Προτύπων & Μηχανική Μάθηση

Τελική Εργασία

Γιώργος Πάνου - Κώστας Σπυρόπουλος – Πέτρος Βαλαχέας

Α’ εξάμηνο

Πειραιάς 2019

Περιεχόμενα

1. Θέμα 1ο – Ιατρικά Συστήματα Ασαφούς Λογικής 2
1.1 Εισαγωγή 2
1.1.1 Υπάρχοντα Συστήματα Ιατρικής Υποστήριξης 3
1.1.2 Υπάρχοντα Συστήματα Ασαφούς Λογικής 3
1.2 Συστήματα Διάγνωσης Ασαφούς Λογικής 4
1.2.1 Δομή του συστήματος 4
1.2.2 Μεθοδολογία 5
1.3 Αναφορές 8
2. Θέμα 2ο – Αναγνώριση Χειρόγραφων Ψηφίων 9
2.1 Εισαγωγή 9
2.1.1 Βασικές Ιδέες Νευρωνικών Δικτύων 9
2.1.2 Σκοπός της εργασίας 9
2.2 Δομή ANN 10
2.2.1 Dataset 10
2.2.2 Δομή NN και Hyperparameters 11
2.3 Λειτουργία ANN 12
2.3.1 Forward Propagation 12
2.3.2 Cost Function 12
2.3.3 Backward Propagation 13
2.4 Εμφάνιση Αποτελεσμάτων 15
2.5 Αποτελέσματα - Συμπεράσματα 19
2.6 Παράρτημα - Κώδικας 19
3. Θέμα 3ο – Hierarchical Clustering 23
3.1 Agglomerative hierarchical clustering 23
3.2 Imports & Βοηθητικές Μέθοδοι 24
3.3 Προετοιμασία δεδομένων 25
3.4 Ιεραρχικό Clustering 28
3.5 Εξαγωγή clusters 28
3.6 Δενδρόγραμμα 31

3.7 Εφαρμογή K-Nearest-Neighbors 33
3.8 Κώδικας Hierarchical Clustering: 36
3.9 Κώδικας KNN: 38
4. Θέμα 4ο - Genetic Travelling Salesman Problem 41
4.1 Εισαγωγή 41
4.1.1 Βασικές Πληροφορίες Γενετικών Αλγορίθμων 41
4.1.2 Σκοπός της εργασίας 41
4.2 Περιγραφή Προβλήματος 42
4.3 Δημιουργία Πληθυσμού (Population) 42
4.4 Υπολογισμός Fitness 42
4.5 Επιλογή Γονέων (Parent Selection) 43
4.6 Διασταύρωση (Crossover) 44
4.7 Μετάλλαξη (Mutation) 46
4.8 Elitism 47
4.9 Αποτελέσματα 47

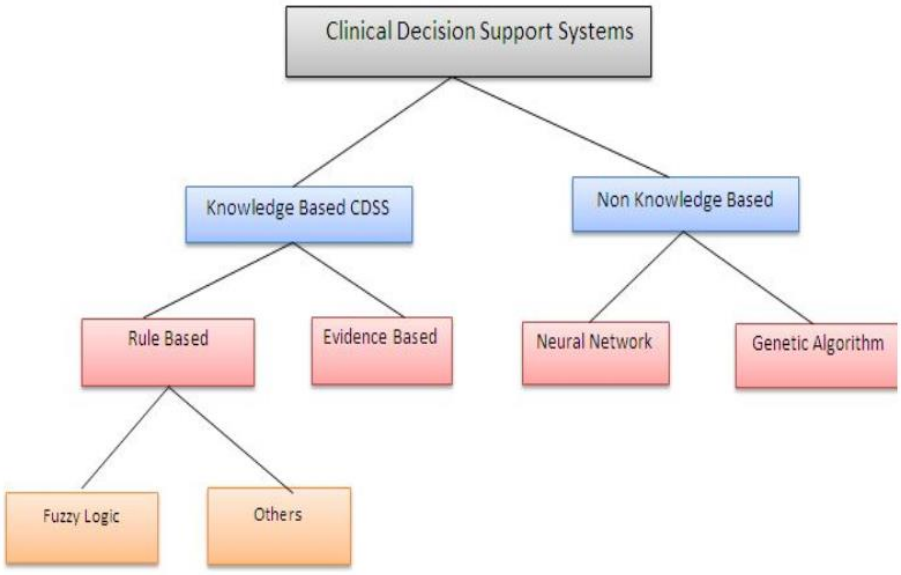
1. Θέμα 1ο – Ιατρικά Συστήματα Ασαφούς Λογικής

1.1 Εισαγωγή

Τα συστήματα Ασαφούς Λογικής βρίσκουν εφαρμογή σε πολλούς τομείς της τεχνολογίας. Το βασικό πλεονέκτημά τους σε σχέση με κοινά συστήματα κανόνων είναι η ευκολία που παρέχουν στον εμπειρογνώμονα να περάσει στο σύστημα τις γνώσεις του καθώς το σύνολο των κανόνων που αποτελεί την βάση για την εξαγωγή των αποτελεσμάτων είναι πολύ μικρότερο. Επίσης για τον ίδιο λόγο επιτρέπουν στο σύστημα να μεταβάλλεται, με μεγαλύτερη ευκολία, ακολουθώντας τις αλλαγές στο γνωστικό πεδίο εφαρμογής.[4]

Στον τομέα της Ιατρικής έχουν αναπτυχθεί χρησιμοποιούνται ήδη από το 1960 ευφυή συστήματα υποστήριξης της διάγνωσης. Τα συστήματα αυτά μπορούν να χωριστούν σε symbolic / non-symbolic ή αλλιώς Knowledge based και Non Knowledge based. Στα Knowledge based συστήματα η αναπαράσταση της γνώσης βρίσκεται σε μορφή που αντικατοπτρίζει ένα μέρος της πραγματικότητας, στα Non Knowledge based η γνώση παράγεται από το μηχάνημα σε μορφή που δεν μπορούμε να την καταλάβουμε καθώς, μέσα από τις διαδικασίες εκπαίδευσης του μηχανήματος, τα δεδομένα που προκύπτουν μπορεί να είναι προβολές των αρχικών δεδομένων σε ανώτερες διαστάσεις.[5]

Τα συστήματα Ασαφούς Λογικής ανήκουν στην πρώτη κατηγορία στην οποία το σύνολο της γνώσης έχει καταχωρηθεί στο σύστημα σε μορφή κανόνων ασαφών συνόλων. Στην Ασαφή Λογική οι τιμές αλήθειας δεν είναι 0 ή 1 αλλά μπορούν να πάρουν οποιαδήποτε τιμή ανάμεσα στις τιμές μεταξύ 0 και 1.



Εικόνα 1 : Κατηγοριοποίηση Συστημάτων Ιατρικής Υποστήριξης

1.1.1 Υπάρχοντα Συστήματα Ιατρικής Υποστήριξης

AAPHelp: de Dombal's system for acute abdominal pain (1972) [6]
Αναπτύχθηκε στο Πανεπιστήμιο του Leeds ώστε να βοηθήσει στην απόφαση για χειρουργική επέμβαση. Για την υλοποίησή του χρησιμοποιήθηκε η Bayesian προσέγγιση και είχε ποσοστό επιτυχίας (91.8%) κατά πολύ μεγαλύτερο από τον πιο έμπειρο της κάθε ομάδας του νοσοκομείου (79.6%)

NTERNIST-I (1974) [6]
Το internetist είναι ένα rule-based expert system που αναπτύχθηκε στο Πανεπιστήμιο του Pittsburgh για την διάγνωση πολύπλοκων προβλημάτων γενικής ιατρικής. Το NTERNIST-I καλύπτει ένα μεγάλο εύρος από τις γνωστές ασθένειες (80%) έχει όμως μεγάλο κόστος συντήρησης και λειτουργίας λόγω του όγκου των δεδομένων που διατηρεί.

DXplain [6],[7]
Το DXplain είναι ένα decision support system που χρησιμοποιεί ένα σύνολο κλινικών ευρημάτων ώστε να παρέχει μια ταξινομημένη λίστα πιθανών διαγνώσεων που μπορεί να ταιριάζουν με τα ευρήματα. Έχει αναπτυχθεί από το Laboratory of Computer Science, Massachusetts General Hospital, Harvard Medical School και μπορεί να προσφερθεί στους ενδιαφερόμενους και ως υπηρεσία.

1.1.2 Υπάρχοντα Συστήματα Ασαφούς Λογικής

Έχουν αναπτυχθεί μέσα στα χρόνια διάφορα συστήματα ασαφούς λογικής για την υποστήριξη της ιατρικής. Ενδεικτικά ακολουθεί μια λίστα ασθενειών για τις οποίες έχει γίνει η χρήση αυτής της τεχνολογίας : [6]

- ASTHMA
- TUBERCULOSIS
- CANCER
- Fetal electrocardiogram
- MRI
- HYPOTHYROIDISM
- HEART DISEASE DIAGNOSIS
- MENIGIOMA
- MENINGITIS
- Malaria

- Prenatal examination
- Medicine DoseDetermination
- Jaundice

1.2 Συστήματα Διάγνωσης Ασαφούς Λογικής

Ένα σύστημα Ασαφούς Λογικής μπορεί να αναπτυχθεί και να χρησιμοποιηθεί στην υποβοήθηση της διάγνωσης. Είναι κατάλληλη για τον συγκεκριμένο τομέα διότι η ασάφειας της ιατρικής γνώσης, όπως αυτή εκφράζεται με τις φυσικές γλώσσες, μπορεί να αναπαρασταθεί μέσα από ένα τυπικό και ποσοτικό φορμαλισμό [2]. Στην ουσία οι Ιατροί περνούν τη εμπειρική γνώση τους στο σύστημα και έπειτα δίνοντας του τα δεδομένα εισόδου παίρνουν μια εκτίμηση σχετικά με την κατάσταση του ασθενούς. Το τελικό αποτέλεσμα μπορεί να ανήκει στην ασαφή λογική δηλαδή να είναι της μορφής: 70% Άσθμα.

Οι Ιατροί έχουν σαν πληροφορία ένα μεγάλο σύνολο δεδομένων από τα οποία δεν μπορεί να εξαχθεί κάποιο άμεσο συμπέρασμα. Θα πρέπει να γίνει μια σύνθεση όλων των παραμέτρων και να παρθεί μια απόφαση ώστε να προκύψει η διάγνωση χωρίς το κάθε μεμονωμένο στοιχείο των δεδομένων να παρέχει ξεκάθαρη πληροφορία για την κατάσταση του ασθενή. Υπάρχει συνάφεια μεταξύ των ασαφών συνόλων και των δεδομένων των ασθενών όπως για παράδειγμα τα αποτελέσματα εξετάσεων και ο συνδυασμός τους με τα συμπτώματα, και την εξέταση του Ιατρού και το κατά πόσο μπορούν να επηρεάσουν την συμμετοχή του ασθενή (σε ποσοστό) σε κάποια κατηγορία (ασθένεια).

Αυτό καθιστά την ασαφή λογική ως ένα εργαλείο εύκολο να αναπτυχθεί και να συντηρηθεί αφού η δημιουργία των κανόνων συμμετοχής στην κάθε κατηγορία μοιάζει πολύ με τον τρόπο που ενεργούν οι Ιατροί κατά την παραδοσιακή διάγνωση[3].

1.2.1 Δομή του συστήματος

Ο πιο συνηθισμένος τρόπος δόμησης ενός συστήματος ασαφούς λογικής μπορεί να αναλυθεί σε 4 διαδοχικά στάδια: [3]

1. **Fuzzification**

   Η μετατροπή της πραγματικής τιμής εισόδου σε μια τιμή ενός ασαφούς συνόλου για να καθοριστεί η τιμή αλήθειας του κάθε κανόνα.

2. **Inference**

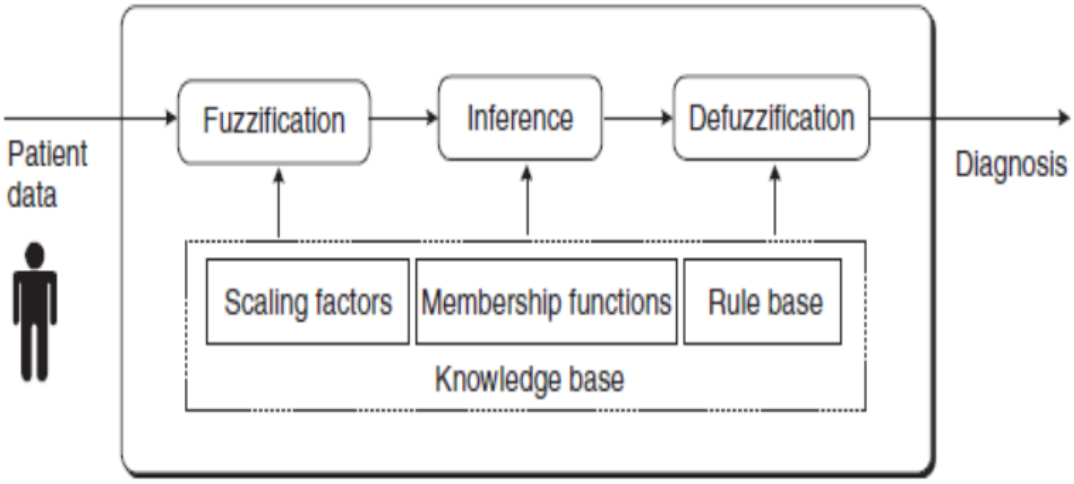
   Καθορίζεται ο βαθμός αλήθειας για τον κάθε κανόνα. Το αποτέλεσμα είναι ένα ασαφές υποσύνολο για κάθε κανόνα της κάθε μεταβλητής εξόδου.

3. **Composition:**

   Όλα τα ασαφή υποσύνολα της κάθε μεταβλητής εξόδου ενώνονται σε ένα υποσύνολο για την κάθε μεταβλητή.

4. **Defuzification:**

   Στην περίπτωση που χρειάζεται μπορεί να επιστραφεί το σύνολο με την μεγαλύτερη τιμή αλήθειας από τα ασαφή σύνολά που αληθεύουν. Για παράδειγμα «καρδιοπάθεια» αν έχει τιμή αλήθειας πάνω από 70%. Με αυτή την διαδικασία όμως χάνεται πληροφορία.

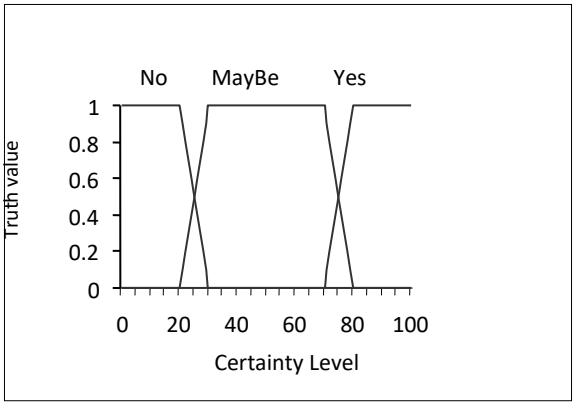


Εικόνα 2 : Ένα σύστημα διάγνωσης Ασαφούς Λογικής

1.2.2 Μεθοδολογία

Μια προσέγγιση για την ανάπτυξη ενός τέτοιου συστήματος είναι να καταχωρηθεί η εμπειρία του Ιατρού σε ένα σύνολο από ασαφείς πίνακες. Έπειτα μπορεί να εφαρμοστεί Inference ώστε να αναπτυχθεί ένα πρόγραμμα που μπορεί να υπολογίσει την βεβαιότητα να πάσχει από μια αρρώστια του συνόλου των αναμενόμενων ασθενειών, ένας ασθενής που παρουσιάζει ένα αριθμό συγκεκριμένων συμπτωμάτων. Η βεβαιότητα αυτή μπορεί να εκφραστεί ως ένας πραγματικό ποσοστό για την κάθε ασθένεια.[1]

Ο κάθε πίνακας καθορίζει το προφίλ μιας ασθένειας. Μπορούν να οριστούν τρία ασαφή σύνολα **Ναι, Ίσως,** και **Όχι** για να αναπαραστίσουν την βεβαιότητα ύπαρξης της κάθε ασθένειας.



Είναι ευθύνη του εμπειρογνώμονα Ιατρού να καθορίσει την τιμή για κάθε χαρακτηριστικό του πίνακα της κάθε ασθένειας. Ένα παράδειγμα πίνακα προφίλ για την γρίπη είναι το εξής:

Attributes **Features**	Very Low	Low	Moderate	High	Very High
Runny nose	No	No	May be	Yes	Yes
Fever	No	May be	Yes	Yes	Yes
Cough	No	May be	Yes	Yes	Yes
Body aches strong	No	May be	Yes	Yes	Yes
Headache	No	May be	Yes	Yes	Yes
Conjunctivitis	No	No	May be	Yes	Yes
Swollen lymph nodes in neck	No	No	May be	May be	May be
Weakness in body	No	No	May be	Yes	Yes
Vomiting	No	No	Yes	May be	May be
Sore throat	No	No	Yes	Yes	Yes
Loss of appetite	No	Maybe	Yes	Yes	Yes
Sneezing	Maybe	Maybe	Yes	Yes	Yes

Στο επόμενο βήμα του Inference εισάγονται τα στοιχεία των εξετάσεων του ασθενή στο σύστημα. Αν τα στοιχεία είναι ποιοτικά εισάγονται στο σύστημα ενώ γίνεται fuzzification σε στοιχεία που είναι μετρήσιμα (όπως π.χ. θερμοκρασία) για να πάρουμε τιμές του τύπου: Πολύ υψηλό,Υψηλό, Μέτριο, Χαμηλό, Πολύ Χαμηλό.

Εάν ορίσουμε ως εξής:

$s[f]$ = ασαφής τιμή για το σύμπτωμα f

r_{ij} = j-οστό συναφές σύμπτωμα i-οστής ασθένειας

$P_{ij}[r_{ij}, v]$ = βεβαιότητα παρουσίας της i-οστής ασθένειας όταν το αντίστοιχο σύμπτωμα έχει ασαφή τιμή v

δ_{ij} = απόφαση διάγνωσης

k_i = συνολικός αριθμός σχετικών χαρακτηριστικών της j-οστής ασθένειας

w_{ij} = βάρος του j-οστού συμπτώματος σε σχέση με την i-οστή ασθένεια.

σ_i = τελική διάγνωση σχετικά με την i-οστή ασθένεια.

Το αντίκτυπο του r_{ij} συμπτώματος της διάγνωσης μπορεί να υπολογιστεί απευθείας από τον πίνακα προφίλ της ασθένειας $P_{ij}[r_{ij}, v]$. Η τιμή του v μπορεί να εξαχθεί από το effect δ_{ij} που ανήκει σε ένα από τα ασαφή σύνολα **Ναι, Ίσως,** και **Όχι**. Η αναπαράστασή του είναι:

$$\delta_{ij} = P_{ij}[r_{ij}, s[r_{ij}]].$$

Αθροίζοντας το αντίκτυπο όλων των σχετικών k_i συμπτωμάτων, η συνολική διάγνωση για την j-οστή ασθένεια μπορεί να υπολογιστεί από τον τύπο:

$$\sigma_i = \left(\sum_{j=1}^{j=k_i} w_{ij}\delta_{ij}\right) / \left(\sum_{j=1}^{j=k_i} w_{ij}\right)$$

Όπου το wi είναι το βάρος που έχει ορίσει ο Ιατρός για το κάθε σύμπτωμα.

1.3 Αναφορές

[1] Methodology for Medical Diagnosis based on Fuzzy Logic, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=2ahUKEwjY3O7u4eDhAhXMKFAKHZjBCdgQFjABegQIAhAC&url=https%3A%2F%2Fwww.kau.edu.sa%2FFiles%2F0052366%2FResearches%2F29055_Med%2520Fuzzy%2520ES.doc&usg=AOvVaw3-mGuXeCowqz60PCYaOT4A

[2] ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ Σημειώσεις Διδασκαλίας Θεμιστοκλής Ν. Παναγιωτόπουλος, https://gunet2.cs.unipi.gr/modules/document/file.php/TMF118/%ce%a3%ce%b7%ce%bc%ce%b5%ce%b9%cf%8e%cf%83%ce%b5%ce%b9%cf%82/notes-ai-msc-ais.pdf

[3] A STUDY ON FUZZY LOGIC AND ITS APPLICATIONS IN MEDICINE, https://acadpubl.eu/hub/2018-119-16/1/145.pdf

[4] Fuzzy Medical Diagnosis, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.9705&rep=rep1&type=pdf

[5] A Survey on the Applications of Fuzzy Logic in Medical Diagnosis, https://pdfs.semanticscholar.org/eafd/ea06efc9160fadfdda90300dc029871a51e4.pdf

[6] A Survey On Fuzzy Logic Applications in Medical Diagnosis https://pdfs.semanticscholar.org/3213/2f2eca0c260ec5c73dd6e73b2891138e98a6.pdf

[7] DXpain : Diagnostic decision support system for general medicine, http://www.mghlcs.org/projects/dxplain

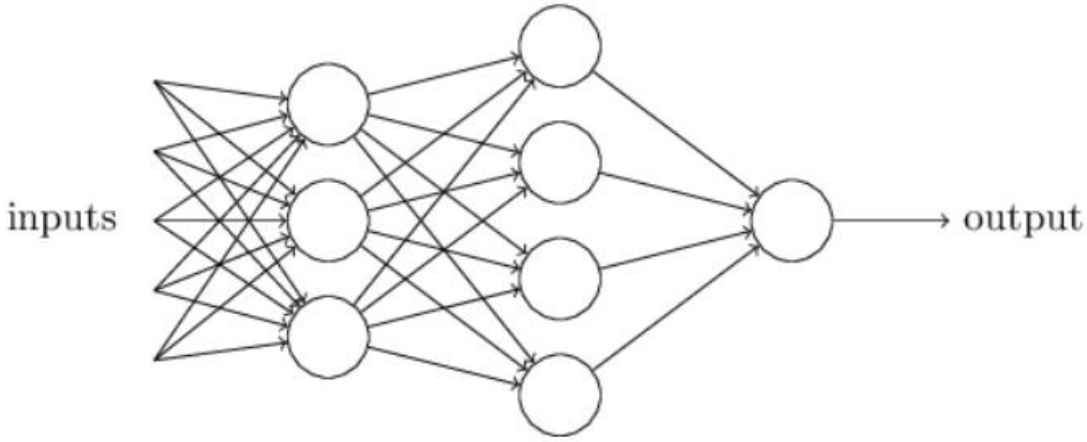
2. Θέμα 2ο – Αναγνώριση Χειρόγραφων Ψηφίων

2.1 Εισαγωγή

2.1.1 Βασικές Ιδέες Νευρωνικών Δικτύων

Η ιδέα των τεχνητών νευρωνικών δικτύων (Artificial Neural Networks) άρχισε να αναπτύσσεται την δεκαετία του ‘50 από τον Frank Rosenblatt, με επιρροές από προηγούμενες έρευνες των Warren McCulloch και Walter Pitts, και αναφέρονταν ως perceptrons. Τα νευρωνικά δίκτυα είναι υπολογιστικά συστήματα εμπνευσμένα από τα βιολογικά νευρωνικά δίκτυα από τα οποία αποτελείται ο εγκέφαλος, και αποτελούν μια αρκετά απλουστευμένη εκδοχή αυτού. Αυτά τα συστήματα “μαθαίνουν” να εκτελούν ενέργειες για τις οποίες δεν έχουν άμεσα προγραμματιστεί, βασιζόμενα σε ένα πλήθος παραδειγμάτων που έχει αρχικά δοθεί.

Τα ANNs αποτελούνται απο επίπεδα νευρώνων που συνδέονται μεταξύ τους με ακμές μεταφέροντας σήματα. Ο κάθε νευρώνας έχει μια παράμετρο που ονομάζεται bias, και κάθε ακμή μια παράμετρο που ονομάζεται βάρος (weight). Αυτές οι παράμετροι αρχικοποιούνται, και στη συνέχεια “ρυθμίζονται” με βάση τα παραδείγματα που δίνουμε στο ANN. Τα παραδείγματα αυτά αποτελούνται από τα δεδομένα εισόδου X και τις ετικέτες Y. Το νευρωνικό δίκτυο δέχεται την είσοδο X ενός απο τα παραδείγματα, συγκρίνει την έξοδο του με την ετικέτα του παραδείγματος που δείχνει τη σωστή έξοδο, και βάσει του σφάλματος “επιστρέφει” πίσω στις παραμέτρους του και τις τροποποιεί έτσι ώστε στο επόμενο παράδειγμα να ελαχιστοποιήσει το σφάλμα του.



Παράδειγμα νευρωνικού δικτύου με 2 επίπεδα

2.1.2 Σκοπός της εργασίας

Στη παρούσα εργασία, σκοπός είναι η δημιουργία ενός νευρωνικού δικτύου το οποίο θα αναγνωρίζει χειρόγραφα ψηφία 0 και 1, χρησιμοποιώντας το dataset usps_mat που μας δόθηκε το οποίο περιέχει 11000 εικόνες για ψηφία απο το 0-9. Για την υλοποίηση του χρησιμοποιήθηκε το Matlab R2019a.

2.2 Δομή ANN

2.2.1 Dataset

Για την εργασία φορτώνουμε το dataset usps_all.mat. Το dataset αυτό είναι ένας 3-διάστατος πίνακας αποτελούμενος από 1100 εικόνες grayscale διαστάσεων 16x16 pixels (256 elements), για κάθε ένα από τα ψηφία 0-9. Για της ανάγκες της εργασίας, χρησιμοποιήσαμε τα data που αντιστοιχούν μόνο στα ψηφία 0 και 1. Στη συνέχεια, διαιρούμε κάθε στοιχείο του πίνακα X με το 255, ώστε όλες οι τιμές να ανήκουν στο εύρος [0,1].

Τέλος, χωρίζουμε το dataset, τον πίνακα X που περιέχει τα δεδομένα και τον πίνακα Y που περιέχει τις ετικέτες των παραδειγμάτων (0/1), σε training set και test set. Το νευρωνικό μας δίκτυο θα εκπαιδευτεί με το training set, και μετά το πέρας της εκπαίδευσής του επαληθεύουμε τα αποτελέσματα χρησιμοποιώντας το test set.

```
%Loading Dataset
load('usps_all.mat');
% Matrix data is 256 x 1100 x 10 , 3 - dimensional array
% 1 and 10 are 1 and 0 digits only
X_data = [data(:,:,1)';data(:,:,10)'];
X = X_data;
X = double(X);
X = X ./ 255;


% X, Y modification
X_train_orig = [X(1:1000,1:256);X(1101:2100,1:256)];
X_test_orig =  [X(1001:1100,1:256);X(2101:2200,1:256)];
X_train_t = X_train_orig';
X_test_t = X_test_orig';
Y_train_orig = [ones(1,NTrain/2),zeros(1,NTrain/2)];
Y_test_orig = [ones(1,NTest/2),zeros(1,NTest/2)];
```

2.2.2 Δομή NN και Hyperparameters

Οι τελικές παράμετροι (hyperparameters) που χρησιμοποιήθηκαν στο συστημά μας βρέθηκαν μετά από αρκετή βελτιστοποίηση και κάνοντας πολλές δοκιμές. Καταλήξαμε στην δημιουργία ενός NN με 2 hidden layers αποτελούμενα απο 4 νευρώνες το κάθε ένα. Το output layer αποτελείται από έναν νευρώνα, ο οποίος δείχνει και το αποτέλεσμα του νευρωνικού δικτύου (0/1). Ο αριθμός των layers, καθώς και ο αριθμός των νευρώνων που έχει το κάθε layer συνήθως βρίσκεται εμπειρικά ανάλογα και με τις ανάγκες του προβλήματος, παίζει όμως σημαντικό ρόλο στην απόδοση του νευρωνικού δικτύου. Περισσότερα layers αλλά και νευρώνες οδηγούν σε καλύτερα αποτελέσματα, αλλά αυτό απαιτεί και περισσότερα δεδομένα για να εκπαιδευτεί το σύστημα καθώς και μεγαλύτερη επεξεργαστική ισχύ, αφού ο αριθμός των παραμέτρων που θα εκπαιδευτούν αυξάνεται. Επίσης σημαντικός παράγοντας είναι και το φαινόμενο του “overfitting”.

Ο αριθμός των epochs, οι επαναλήψεις δηλαδή που θα “περάσουν” τα δεδομένα από το σύστημά μας, επιλέχτηκαν στις 700. Σε κάθε epoch το σφάλμα θα μειώνεται, άρα λιγότερες επαναλήψεις οδηγούν σε μη επαρκή μάθηση του συστήματος μας πάνω στα δεδομένα.

Τέλος, θέτουμε την τιμή alpha = 0.4, η οποία αποτελεί το learning rate στο οποίο θα αναφερθούμε αργότερα, αρχικοποιούμε τα weights με τυχαίες τιμές και τα biases με μηδενικά και δημιουργούμε και έναν πίνακα cost_overall ώστε να αποθηκεύουμε το σφάλμα σε κάθε εποχή για να αναπαρασταθεί γραφικά.

```
% Initialize Parameters
NTrain = 2000;
NTest = 200;
epochs = 700;
alpha = 0.4;
n_x = 256;
n_h = 4;
n_h2 = 4;
n_y = 1;
% w, b Initialization
w1 = rand(n_h,n_x)*0.01;
b1 = zeros(n_h,1);
w2 = rand(n_h2,n_h)*0.01;
b2 = zeros(n_h2,1);
w3 = rand(n_y,n_h2)*0.01;
b3 = zeros(n_y,1);
cost_overall = zeros(1,epochs);
```

2.3 Λειτουργία ANN

2.3.1 Forward Propagation

Σε κάθε epoch, τα δεδομένα εισόδου τροφοδοτούνται στο δίκτυο με εμπρόσθια κατεύθυνση, κάθε layer δέχεται τα δεδομένα και τα επεξεργάζεται μέσω της συνάρτησης ενεργοποίησης. Έπειτα αυτα τα δεδομένα προωθούνται στο επόμενο layer.

Η συνάρτηση για κάθε νευρώνα για forward propagation είναι η εξής:

$$\textbf{Feed Forward: } Z = W * X + b \qquad (1)$$

Μετά από την συνάρτηση (1), τα δεδομένα περνούν μέσα από μια συνάρτηση ενεργοποίησης (activation function). Υπάρχουν αρκετές συναρτήσεις ενεργοποίησης που συναντώνται συχνά στα νευρωνικά δίκτυα, όπως η πιο κοινή sigmoid function, η tanh function, ReLu, Leaked ReLu κτλ. Στη δική μας περίπτωση χρησιμοποιούμε την tanh για τα δύο hidden layers και την sigmoid για το output layer.

$$\textbf{Sigmoid Function: } A = \frac{1}{1+e^{-z}} \qquad (2)$$

$$\textbf{Tanh Function: } A = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \qquad (3)$$

Χρησιμοποιήσαμε δυο διαφορετικές συναρτήσεις ενεργοποίησης, στα hidden layers και στο output layer διότι μετά από πολλές δοκιμές αυτή η τεχνική βοήθησε σε καλύτερα αποτελέσματα.

```
% Forward Propagation
Z1 = w1*X_train_t + b1;
A1 = tanh(Z1);
Z2 = w2*A1 + b2;
A2 = tanh(Z2);
Z3 = w3*A2 + b3;
A3 = 1./(1 + exp(-Z3));
```

2.3.2 Cost Function

Το αποτέλεσμα του δικτύου, ο πίνακας δηλαδή A3, είναι η εκτίμηση του νευρωνικού μας δικτύου για το τι είναι οι εικόνες που δέχθηκε σαν είσοδο. Αυτή η εκτίμηση πρέπει να συγκριθεί με την επιθυμητή έξοδο.

```
% Cost Function
logprobs = log(A3')*Y_train_orig + (1-Y_train_orig)*(log(1-A3'));
cost = -(1/NTrain).*(sum(logprobs));
cost_overall(i) = mean(cost);
```

Ομοίως με τις συναρτήσεις ενεργοποίησης, υπάρχουν διάφορες συναρτήσεις κόστους (cost function ή error function ή loss function). Μερικές από αυτές είναι η mean squared error και η cross-entropy cost function. Εμείς χρησιμοποιήσαμε την cross-entropy cost function σαν μια πιο αξιόπιστη αλλά παράλληλα απλή συνάρτηση κόστους.

Cross-entropy cost function: L = - (y * log(p) + (1 - y) * log(1 - p)) (4)

2.3.3 Backward Propagation

Σε αυτό το σημείο, το δίκτυο μας θα πρέπει να μάθει, δηλαδή να ρυθμίσει τα βάρη και τα biases του νευρωνικού ώστε στην επόμενη epoch το σφάλμα μας να είναι μικρότερο, να έχει δηλαδή μεγαλύτερο ποσοστό επιτυχίας το δίκτυό μας ως πρός την εκτίμηση του για τα δεδομένα εισόδου. Αυτό γίνεται μέσω του Back Propagation, όπου υπολογίζονται οι μερικές παράγωγοι των παραμέτρων μας με την τεχνική της οπισθοδιάδοσης.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

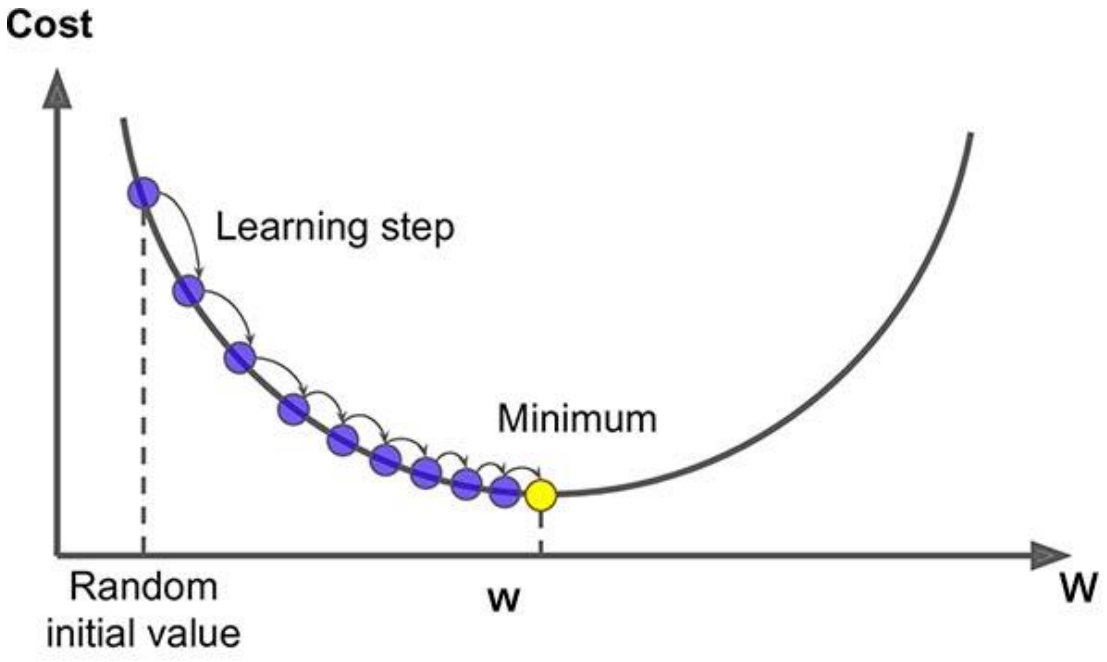
Μερική παράγωγος του σφάλματος ως προς το βάρος

```
% Backward Propagation
dZ3 = A3 – Y_train_orig;
dw3 = (1/Ntrain) * (dZ3*A2’);
db3 = (1/Ntrain) * (sum(dZ3,2));
dZ2 = ((w3’)*dZ3) .* (1 – A2.^2);
dw2 = (1/Ntrain) * (dZ2*A1’);
db2 = (1/Ntrain) * (sum(dZ2,2));
dZ1 = ((w2’)*dZ2) .* (1 – A1.^2);
dw1 = (1/Ntrain) * (dZ1*X_train_t’);
db1 = (1/Ntrain) * (sum(dZ1,2));
```

Για να εκπαιδευτεί το δίκτυο, κάνουμε χρήση του αλγορίθμου Gradient Descent. Ο αλγόριθμος Gradient Descent μας δείχνει το πως να αλλάξουμε τις παραμέτρους μας ώστε να ελαχιστοποιηθεί το σφάλμα, βρίσκοντας το ελάχιστο τοπικό ακρότατο μιας συνάρτησης, δηλαδή της συνάρτησης κόστους. Εδώ χρησιμοποιούμε και την προαναφερθείσα παράμετρο alpha, η οποία ελέγχει το πόσο γρήγορα θα εκπαιδεύεται το δίκτυο.

```
% Update parameters
w1 = w1 - alpha * dw1;
b1 = b1 - alpha * db1;
w2 = w2 - alpha * dw2;
b2 = b2 - alpha * db2;
w3 = w3 - alpha * dw3;
b3 = b3 - alpha * db3;
```

Ανάλογα με το πρόσημο της παραγώγου σε κάθε epoch, αλλάζει και η κάθε παράμετρος και συνεπάγεται και η μείωση του σφάλματος.



Μείωση του κόστους στο τοπικό ελάχιστο της συνάρτησης σφάλματος παραμετροποιώντας το βάρος (w) με χρήση του αλγορίθμου gradient descent

Επίσης, κρίσιμη είναι και η επιλογή του learning_rate (alpha), καθώς μικρή τιμή του alpha προκαλεί αργή εκπαίδευση του δικτύου, ενώ μεγάλο alpha μπορεί να ξεπεράσει την τιμή του ελάχιστου και το σφάλμα να εκτοξευθεί.

2.4 Εμφάνιση Αποτελεσμάτων

Στο τέλος κάθε epoch, θέλουμε να δούμε την μείωση του σφάλματος. Επειδή όμως είναι πολλές, εμφανίζουμε το κόστος κάθε 100 epochs.

```
% Display Cost
cost = squeeze(cost);
if mod(i,100) == 0
      costdisp = ['Cost after iteration ', num2str(i), ': ',
num2str(mean(cost))];
     disp(costdisp);
end
```

Μετά το τέλος της εκπαίδευσης, για να δούμε την εκτίμηση του δικτύου μας για κάθε παράδειγμα εισόδου, εκτελούμε την feedforward διαδικασία, χωρίς αυτή την φορά να εκπαιδεύσουμε τις παραμέτρους μας. Εάν η τιμή της εκτίμησης είναι 0.5 ή χαμηλότερη, τότε η εκτίμηση του νευρωνικού είναι ότι η εικόνα είναι 0, αλλίως η εικόνα είναι 1.

Ομοίως κάνουμε και για το training set και για το test set. Δίνουμε μεγαλύτερη σημασία στις εκτιμήσεις του δικτύου πάνω στο test set, καθώς αυτά είναι δεδομένα που το δίκτυο δεν τα έχει ξαναδει.

```
% Train Prediction
Y_prediction_train = zeros(1,NTrain);

Z1 = w1*X_train_t + b1;
A1 = tanh(Z1);
Z2 = w2*A1 + b2;
A2 = tanh(Z2);
Z3 = w3*A2 + b3;
A3 = 1./(1 + exp(-Z3));

for i = 1:length(A3)

    if A3(1,i) <= 0.5
        Y_prediction_train(1,i) = 0;
    else
        Y_prediction_train(1,i) = 1;
    end
end

% Test Prediction
Y_prediction_test = zeros(1,NTest);

Z1 = w1*X_test_t + b1;
A1 = tanh(Z1);
Z2 = w2*A1 + b2;
```

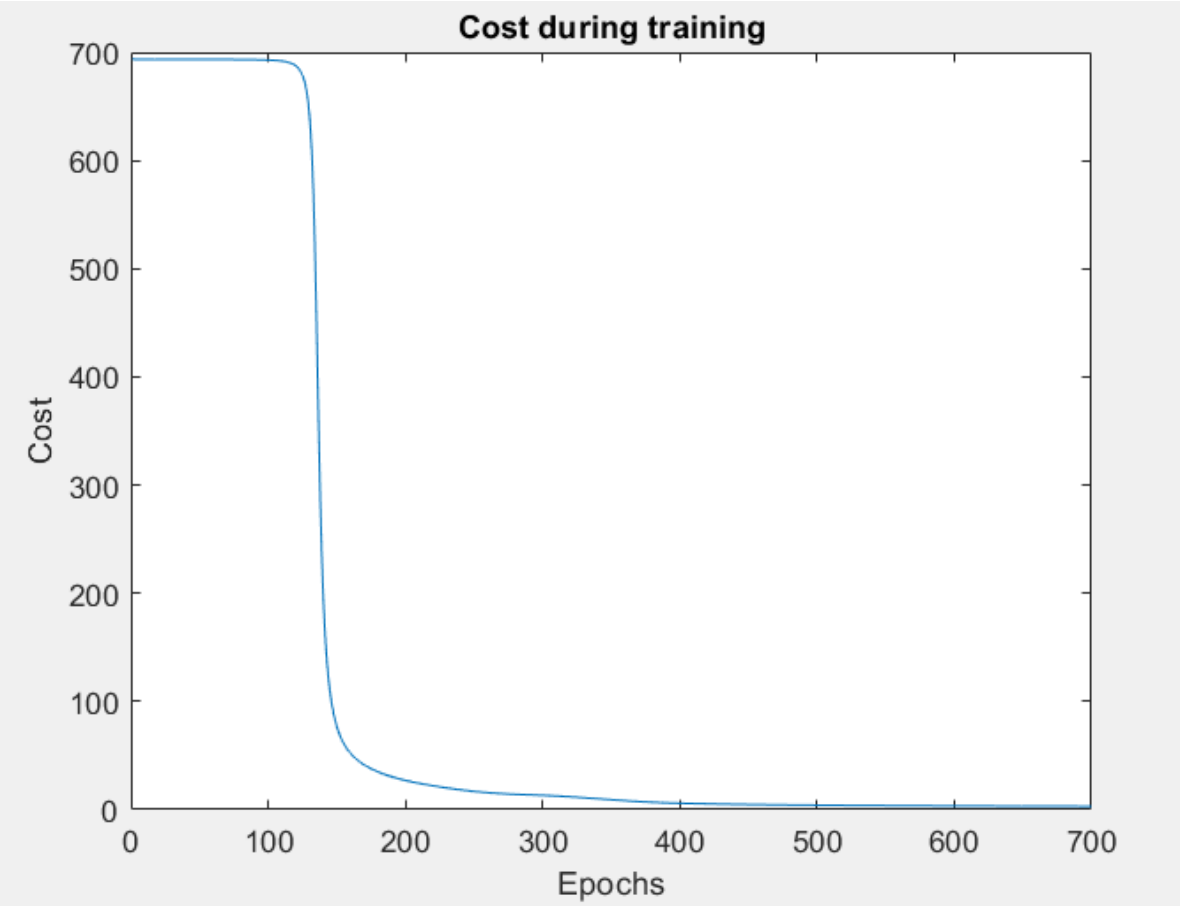
```
A2 = tanh(Z2);
Z3 = w3*A2 + b3;
A3 = 1./(1 + exp(-Z3));

for i = 1:length(A3)

    if A3(1,i) <= 0.5
        Y_prediction_test(1,i) = 0;
    else
        Y_prediction_test(1,i) = 1;
    end
end
```

Παρακάτω παρουσιάζουμε τα ποσοστά επιτυχίας του νευρωνικού δικτύου για κάθε dataset και εμφανίζουμε γραφικά το σφάλμα κατά την εκπαίδευση του νευρωνικού.

```
%Display Train and Test Accuracy
train_accuracy = 100 - (abs(mean(Y_prediction_train) - mean(Y_train_orig))) *
100;
test_accuracy = 100 - (abs(mean(Y_prediction_test) - mean(Y_test_orig))) *
100;
show1 = ['Train accuracy: ', num2str(train_accuracy)];
show2 = ['Test accuracy: ', num2str(test_accuracy)];
disp(newline);
disp(show1);
disp(show2);
figure;
plot(cost_overall);
title('Cost during training');
xlabel('Epochs');
ylabel('Cost');
```

Τα αποτελέσματα του δικτύου μας είναι τα εξής:

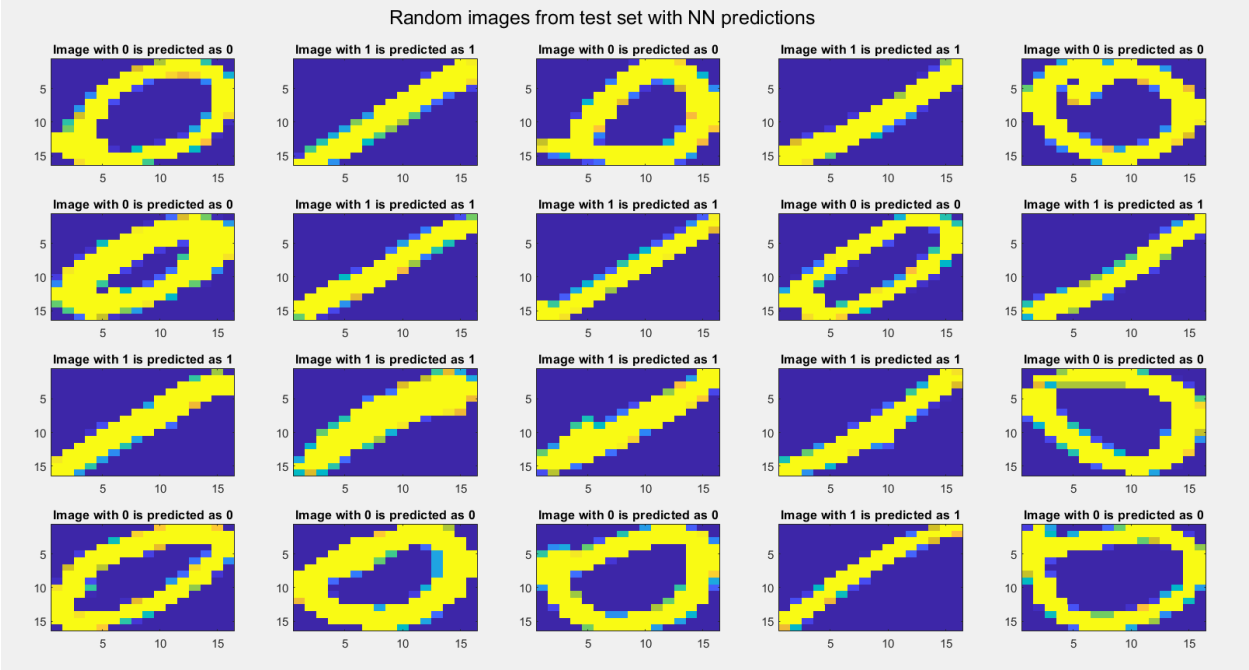
```
Cost after iteration 100: 692.8443
Cost after iteration 200: 27.0508
Cost after iteration 300: 13.2551
Cost after iteration 400: 5.9436
Cost after iteration 500: 4.1676
Cost after iteration 600: 3.4979
Cost after iteration 700: 3.143


Train accuracy: 100
Test accuracy: 99.5
```



Γραφική παράσταση σφάλματος κατα την διάρκεια της εκπαίδευσης

Τέλος, εμφανίζουμε τυχαίες εικόνες από το test dataset μας με τίτλο την πρόβλεψη που έκανε για αυτές το νευρωνικό μας δίκτυο μετά την εκπαίδευσή του.

```
%Show random images from test set with predictions
pixels = sqrt(n_x);
img = zeros(pixels, pixels);
X_img =  [X_data(1001:1100,1:256);X_data(2101:2200,1:256)];
figure;
for k = 1:20
    r = randi([1 NTest], 1);
    for i = 1:pixels
        for j = 1:pixels
            img(i, j) = X_img(r, ((i-1)*16)+j);
        end
    end
    subplot(4,5,k)
    image(img)
    show3 = ['Image with ', num2str(Y_test_orig(1,r)), ' is predicted as ',
num2str(Y_prediction_test(1,r))];
    title(show3)
end
sgtitle('Random images from test set with NN predictions')
```



Τυχαίες εικόνες από το test set με το prediction του νευρωνικού δικτύου

2.5 Αποτελέσματα - Συμπεράσματα

Παρατηρούμε ότι κάθε 100 epochs, το κόστος μείωνεται, ειδικά τις πρωτες 200 όπου εκεί μειώνεται δραματικά. Τελικά, έχουμε επιτυχία 100% στα δεδομένα του training set και 99.5% στα δεδομένα του test set. Αυτό επιτεύχθηκε μετά από αρκετές τροποποιήσεις του δικτύου, αφού στην πρώτη προσπάθεια σε ένα νευρωνικό δίκτυο ενός layer 5 νευρώνων πετυχαίναμε επιτυχία στο test set της τάξης του 60%.

Η διαδικασία εύρεσης των κατάλληλων τιμών για τις μη-εκπαιδεύσιμες παραμέτρους του συστήματος μας ήταν το πιο απαιτητικό κομμάτι για την υλοποίηση του νευρωνικού δικτύου. Ο αριθμός των epochs, η παράμετρος alpha αλλά και ο αριθμός των layer και των νευρώνων/layer ήταν τα πιο καθοριστικά σημεία για την σωστή λειτουργία του δικτύου.

2.6 Παράρτημα - Κώδικας

```
clear all, clc

%Loading Dataset
load('usps_all.mat');
% Matrix data is 256 x 1100 x 10 , 3 - dimensional array
% 1 and 10 are 1 and 0 digits only
X_data = [data(:,:,1)';data(:,:,10)'];
X = X_data;
X = double(X);
X = X ./ 255;

% Initialize Parameters
NTrain = 2000;
NTest = 200;
epochs = 2000;
alpha = 0.4;
n_x = 256;
n_h = 4;
n_h2 = 4;
n_y = 1;

% X, Y modification
X_train_orig = [X(1:1000,1:256);X(1101:2100,1:256)];
X_test_orig =  [X(1001:1100,1:256);X(2101:2200,1:256)];
X_train_t = X_train_orig';
X_test_t = X_test_orig';
Y_train_orig = [ones(1,NTrain/2),zeros(1,NTrain/2)];
Y_test_orig = [ones(1,NTest/2),zeros(1,NTest/2)];

% w, b Initialization
w1 = rand(n_h,n_x)*0.01;
b1 = zeros(n_h,1);
w2 = rand(n_h2,n_h)*0.01;
```

```
b2 = zeros(n_h2,1);
w3 = rand(n_y,n_h2)*0.01;
b3 = zeros(n_y,1);
cost_overall = zeros(1,epochs);


% Forward Propagation, Cost Function and Backward Propagation
for i = 1:epochs

    % Forward Propagation
    Z1 = w1*X_train_t + b1;
    A1 = tanh(Z1);
    Z2 = w2*A1 + b2;
    A2 = tanh(Z2);
    Z3 = w3*A2 + b3;
    A3 = 1./(1 + exp(-Z3));

    % Cost Function
    logprobs = log(A3')*Y_train_orig + (1-Y_train_orig)*(log(1-A3'));
    cost = -(1/NTrain).*(sum(logprobs));
    cost_overall(i) = mean(cost);


    % Backward Propagation
    dZ3 = A3 - Y_train_orig;
    dw3 = (1/NTrain) * (dZ3*A2');
    db3 = (1/NTrain) * (sum(dZ3,2));
    dZ2 = ((w3')*dZ3) .* (1 - A2.^2);
    dw2 = (1/NTrain) * (dZ2*A1');
    db2 = (1/NTrain) * (sum(dZ2,2));
    dZ1 = ((w2')*dZ2) .* (1 - A1.^2);
    dw1 = (1/NTrain) * (dZ1*X_train_t');
    db1 = (1/NTrain) * (sum(dZ1,2));

    % Update parameters
    w1 = w1 - alpha * dw1;
    b1 = b1 - alpha * db1;
    w2 = w2 - alpha * dw2;
    b2 = b2 - alpha * db2;
    w3 = w3 - alpha * dw3;
    b3 = b3 - alpha * db3;

    % Display Cost
    cost = squeeze(cost);
    if mod(i,100) == 0
        costdisp = ['Cost after iteration ', num2str(i), ': ',
num2str(mean(cost))];
        disp(costdisp);
    end
end
```

```
% Train Prediction
Y_prediction_train = zeros(1,NTrain);

Z1 = w1*X_train_t + b1;
A1 = tanh(Z1);
Z2 = w2*A1 + b2;
A2 = tanh(Z2);
Z3 = w3*A2 + b3;
A3 = 1./(1 + exp(-Z3));

for i = 1:length(A3)

    if A3(1,i) <= 0.5
        Y_prediction_train(1,i) = 0;
    else
        Y_prediction_train(1,i) = 1;
    end
end

% Test Prediction
Y_prediction_test = zeros(1,NTest);

Z1 = w1*X_test_t + b1;
A1 = tanh(Z1);
Z2 = w2*A1 + b2;
A2 = tanh(Z2);
Z3 = w3*A2 + b3;
A3 = 1./(1 + exp(-Z3));

for i = 1:length(A3)

    if A3(1,i) <= 0.5
        Y_prediction_test(1,i) = 0;
    else
        Y_prediction_test(1,i) = 1;
    end
end


%Display Train and Test Accuracy
train_accuracy = 100 - (abs(mean(Y_prediction_train) - mean(Y_train_orig))) *
100;
test_accuracy = 100 - (abs(mean(Y_prediction_test) - mean(Y_test_orig))) *
100;
show1 = ['Train accuracy: ', num2str(train_accuracy)];
show2 = ['Test accuracy: ', num2str(test_accuracy)];
disp(newline);
disp(show1);
disp(show2);
```

```
figure;
plot(cost_overall);
title('Cost during training');
xlabel('Epochs');
ylabel('Cost');


%Show 20 random images from test set with predictions
pixels = sqrt(n_x);
img = zeros(pixels, pixels);
X_img =  [X_data(1001:1100,1:256);X_data(2101:2200,1:256)];
figure;
for k = 1:20
    r = randi([1 NTest], 1);
    for i = 1:pixels
        for j = 1:pixels
            img(i, j) = X_img(r, ((i-1)*16)+j);
        end
    end
    subplot(4,5,k)
    image(img)
    show3 = ['Image with ', num2str(Y_test_orig(1,r)), ' is predicted as ',
num2str(Y_prediction_test(1,r))];
    title(show3)
end
sgtitle('Random images from test set with NN predictions')
```

3. Θέμα 3ο – Hierarchical Clustering

3.1 Agglomerative hierarchical clustering

Το Hierarchical clustering αποτελεί μια γενικότερη οικογένεια αλγορίθμων συσταδοποίησης οι οποίοι δημιουργούν εμφωλευμένες συστάδες είτε συγχωνεύοντας τις ή διασπώντας τις ακολουθιακά. Αυτή η ιεραρχία αναπαρίσταται ως δέντρο ή δενδρόγραμμα. Η ρίζα του δέντρου είναι η μοναδική συστάδα που συγκεντρώνει όλα τα δείγματα, τα φύλλα είναι οι μοναδικές συστάδες με μόνο ένα μέλος.

Το Agglomerative clustering είναι μια προσέγγιση από κάτω προς τα πάνω: κάθε παρατήρηση αποτελεί μια αυτόνομη συστάδα και ζεύγη συστάδων συγχωνεύονται όσο μετακινούμαστε προς τα πάνω της ιεραρχίας.

Για να αποφασίσουμε ποιες συστάδες πρέπει να συγχωνευτούν σε κάθε βήμα απαιτείται να οριστεί ένα μέτρο "ανομοιότητας" μεταξύ δύο παρατηρήσεων. Στις περισσότερες μεθόδους ιεραρχικής συσταδοποίησης αυτό επιτυγχάνεται εφαρμόζοντας μια κατάλληλη μέτρηση της απόστασης μεταξύ των παρατηρήσεων και τον καθορισμό ενός κριτηρίου συγχώνευσης (linkage criterion) ώστε να παρθεί η απόφαση συγχώνευσης των συστάδων.

Οι μέθοδοι που χτίζουν τα clusters από κάτω προς τα πάνω παράγουν διαφορετικά αποτελέσματα από τις αντίστροφες μεθόδους διότι στις τελευταίες ο ορισμό των αρχικών συστάδων επιλέγεται συνήθως τυχαία ή με ανθρώπινη παρέμβαση κάτι που σημαίνει ότι δεν είναι ντετερμινιστικές. Έτσι με τις μεθόδους της δεύτερης κατηγορίας μπορεί να πάρει κανείς διαφορετικά αποτελέσματα σε κάθε εκτέλεση.

Συνοψίζοντας στις από πάνω προς τα κάτω μεθόδους υπάρχει περίπτωση τα αρχικά σημεία/clusters μπορούν να επηρεάσουν τη σύνθεση των clusters σε κάθε βήμα του αλγορίθμου με έναν παράγοντα εξωγενή. Αντίθετα στις από κάτω προς τα πάνω μεθόδους το κάθε σημείο συμμετέχει ισότιμα στην σύνθεση των μελών του κάθε cluster σε κάθε βήμα και το αποτέλεσμα εξαρτάται μόνο από την ανομοιότητα των σημείων ("γεωμετρία" ή κάποια άλλη μετρική) και τον αλγόριθμο.

Δεδομένης και της φύσης του προβλήματος που θέλουμε να λύσουμε, αποφασίσαμε να εφαρμόσουμε μεθόδους από κάτω προς τα πάνω. Αυτό διότι δεν είχαμε στη διάθεσή μας πληροφορίες που να μας προδίδουν την σύνθεση των clusters εκ των προτέρων ώστε να εισάγουμε αυτή την γνώση ως ένα "educated guess".

Σε αυτή την εργασία παρουσιάζονται αποτελέσματα χρήσης τόσο της ευκλείδειας απόστασης των παρατηρήσεων όσο και άλλες μορφές μέτρησης της ανομοιότητας και αναλύονται τα αποτελέσματά τους πάνω στα δεδομένα που μας δόθηκαν.

Τα διαφορετικά αποτελέσματα που προκύπτουν από το κάθε κριτήριο παρουσιάζονται αναλυτικά.

Χρησιμοποιήσαμε την βιβλιοθήκη "The SciPy library" της python ως πλατφόρμα για να εφαρμόσουμε το clustering στα δεδομένα: https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html

Σχόλια σχετικά με τους αλγόριθμους που υλοποιεί και την αντίστοιχη πολυπλοκότητα της μεθόδου linkage της βιβλιοθήκης:

- For method ‘single’ an optimized algorithm based on minimum spanning tree is implemented. It has time complexity n^2
- For methods ‘complete’, ‘average’, ‘weighted’ and ‘ward’ an algorithm called nearest-neighbors chain is implemented. It also has time complexity n^2 . For other methods a naive algorithm is implemented with n^3 time complexity. All algorithms use n^2 memory.
- Methods ‘centroid’, ‘median’ and ‘ward’ are correctly defined only if Euclidean pairwise metric is used. If y is passed as precomputed pairwise distances, then it is a user responsibility to assure that these distances are in fact Euclidean, otherwise the produced result will be incorrect.

Για την είσοδο των δεδομένων έχει χρησιμοποιηθεί η βιβλιοθήκη pandas και για την απεικόνιση η βιβλιοθήκη matplotlib. Το παρόν document έχει παραχθεί με την χρήση του Jupyter Notebook.

3.2 Imports & Βοηθητικές Μέθοδοι

Στην αρχή έχουμε τα imports που πρέπει να γίνουν και δύο βοηθητικές συναρτήσεις για τον συνεπή χρωματισμό των συστάδων, του τελικού αποτελέσματος, τόσο στο δενδρόγραμμα όσο και στην απεικόνιση σε 2 διαστάσεις:

```
import pandas as pd
import xlsxwriter
import matplotlib.pyplot as plt
import numpy

from scipy.cluster.hierarchy import dendrogram, linkage,leaders
from scipy.cluster import hierarchy
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import inconsistent


def binSearchClusterNode(node, nodeID):
  #print(node.get_id(),nodeID)
  found = False
  if node.is_leaf() or node.get_id() < nodeID:
     return None,False

  if nodeID == node.get_id():
    # print("Found it")
     return node,True

  nodeL=node.get_left()
  nodeR=node.get_right()
```

```
  if nodeL.get_id() == nodeID:
     node,found = binSearchClusterNode(nodeL,nodeID)
  elif nodeR.get_id() == nodeID:
     node,found = binSearchClusterNode(nodeR, nodeID)
  else:
     node,found = binSearchClusterNode(nodeL,nodeID)
     if found == False:
        node,found = binSearchClusterNode(nodeR, nodeID)

  return node,found


def getLeafIDFromTreeNode(rootnode,nodelist,rootID):
  # print("BFS--")
  node,found=binSearchClusterNode(rootnode,rootID)
  # print(node.get_id())
  while not node.is_leaf():
     node = node.get_left()

  # print(node.get_id())

  return node.get_id()
```

3.3 Προετοιμασία δεδομένων

Πρώτα φορτώνουμε το dataset από το excel που μας δόθηκε σε ένα dataframe με την χρήση του pandas και προσθέτουμε αυθαίρετα ονομασίες στις συστάδες χαρακτηριστικών

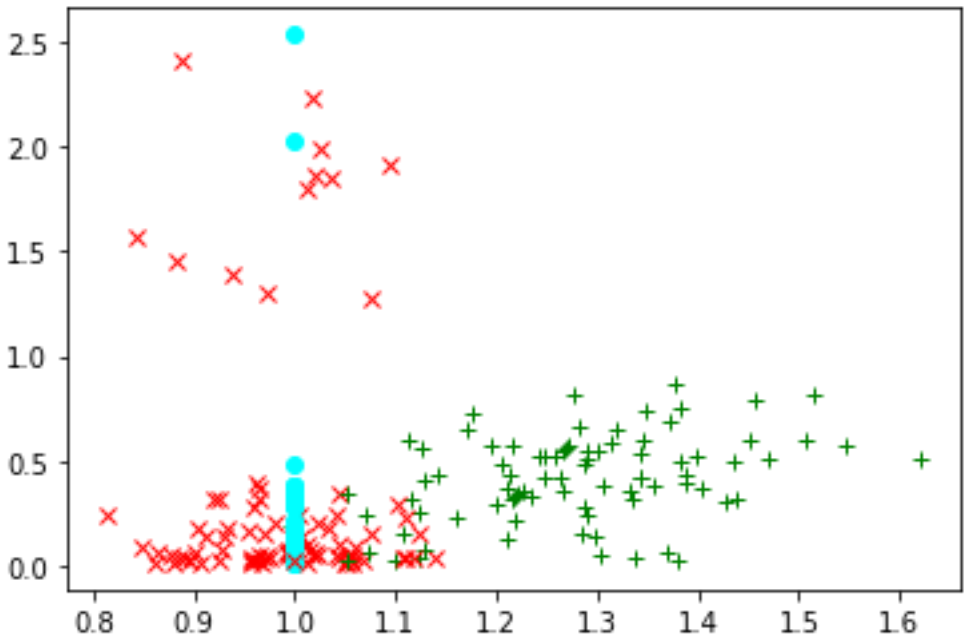
```
pd.options.mode.chained_assignment = None  # default='warn'
df=pd.read_excel("features.xls",header=1,usecols=range(1,84),skiprows="1").dropna()

df=df.reset_index(drop=True)
print(df)

feature1=df.loc[0:2]
feature1.rename(index={0:'Ουδέτερη',1:'Χαρά',2:'Έκπληξη"}, inplace=True)
feature2=df.loc[3:5]
feature2=feature2.reset_index(drop=True)
feature2.rename(index={0:'Ουδέτερη',1:'Χαρά',2:'Έκπληξη"}, inplace=True)

print("Feature1:",feature1)
print("Feature2:",feature2)


     1       2       3     4     5       6       7  \
0 1.000000 1.000000       1 1.00000 1.00000 1.000000 1.000000
```

```
1 1.012200 0.956440     1.045 0.93299 1.01620 1.023800 0.968890
2 1.209400 1.124000   1.3142 1.21760 1.21000 1.341500 1.386900
3 0.099980 0.069767 0.046499 0.09972 0.11891 0.048780 0.046399
4 0.099986 0.037736   0.09992 0.18515 2.23120 0.055546 0.149140
5 0.371990 0.263070   0.59346 0.33918 0.13459 0.428400 0.399350

    8       9      10 ...      74      75      76      77 \
0 1.00000 1.000000 1.00000 ... 1.00000 1.000000 1.000000 1.000000
1 0.93921 1.018800 1.07610 ... 0.81472 0.901750 0.981470 1.014000
2 1.22120 1.403700 1.22600 ... 1.17600 1.262900 1.318100 1.304000
3 0.08854 0.047606 0.30200 ... 0.28443 0.102560 0.074977 0.326460
4 1.39560 1.860000 0.16129 ... 0.24469 0.060495 0.204040 0.072715
5 0.34023 0.370820 0.36214 ... 0.72242 0.418480 0.645940 0.052559

    78      79      80      81      82      83
0 1.00000 1.000000 1.00000 1.000000 1.00000 1.000000
1 0.84192 1.022300 1.03410 0.887570 0.96373 0.998150
2 1.12660 1.212700 1.33550 1.305800 1.45750 1.291100
3 0.09802 0.095211 0.08000 0.060908 0.10226 0.108670
4 1.57070 0.203390 0.18367 2.411600 0.37209 0.032787
5 0.56081 0.436650 0.31557 0.388740 0.78644 0.242450

[6 rows x 83 columns]
Feature1:            1      2      3      4      5      6      7      8  \
Ουδέτερη 1.0000 1.00000      1 1.00000 1.0000 1.0000 1.00000 1.00000
Χαρά    1.0122 0.95644   1.045 0.93299 1.0162 1.0238 0.96889 0.93921
Έκπληξη   1.2094 1.12400 1.3142 1.21760 1.2100 1.3415 1.38690 1.22120

         9      10 ...      74      75      76    77      78 \
Ουδέτερη 1.0000 1.0000 ... 1.00000 1.00000 1.00000 1.000 1.00000
Χαρά    1.0188 1.0761 ... 0.81472 0.90175 0.98147 1.014 0.84192
Έκπληξη   1.4037 1.2260 ... 1.17600 1.26290 1.31810 1.304 1.12660

         79    80      81      82      83
Ουδέτερη 1.0000 1.0000 1.00000 1.00000 1.00000
Χαρά    1.0223 1.0341 0.88757 0.96373 0.99815
Έκπληξη   1.2127 1.3355 1.30580 1.45750 1.29110

[3 rows x 83 columns]
Feature2:            1       2       3      4      5       6       7  \
Ουδέτερη 0.099980 0.069767 0.046499 0.09972 0.11891 0.048780 0.046399
Χαρά    0.099986 0.037736   0.09992 0.18515 2.23120 0.055546 0.149140
Έκπληξη   0.371990 0.263070   0.59346 0.33918 0.13459 0.428400 0.399350

          8       9      10 ...      74      75      76 \
Ουδέτερη 0.08854 0.047606 0.30200 ... 0.28443 0.102560 0.074977
Χαρά    1.39560 1.860000 0.16129 ... 0.24469 0.060495 0.204040
Έκπληξη   0.34023 0.370820 0.36214 ... 0.72242 0.418480 0.645940
```

```
        77     78     79     80     81     82     83
Ουδέτερη 0.326460 0.09802 0.095211 0.08000 0.060908 0.10226 0.108670
Χαρά    0.072715 1.57070 0.203390 0.18367 2.411600 0.37209 0.032787
Έκπληξη  0.052559 0.56081 0.436650 0.31557 0.388740 0.78644 0.242450

[3 rows x 83 columns]
```

Σε αυτό το σημείο μετασχηματίζουμε τα στοιχεία είσοδού σε δομή δεδομένων ώστε να είναι έτοιμα για το clustering (rawData) και έπειτα δημιουργούμε μια απεικόνιση των δεδομένων σε γράφημα 2 διαστάσεων, την κάθε συστάδα με διαφορετικό χρώμα, σύμφωνα με τα στοιχεία εισόδου.

```
data=[]
rawData=[]
for personID, col in feature1.iteritems():
  #print("person: ",personID)
  for cluster, element in enumerate(col):
     data.append([element, feature2[personID][cluster],cluster,personID])
     rawData.append([element, feature2[personID][cluster]])

data=numpy.array(data,dtype = object)
rawData=numpy.array(rawData)

mapping= { 0: ("cyan", "o"), 1: ("red", "x"), 2: ("green", "+") }

for element in data:
  plt.plot(element[0],element[1],c=mapping[element[2]][0], marker=mapping[element[2]][1])
plt.show()
```



3.4 Ιεραρχικό Clustering

Αυτή η συνάρτηση "τρέχει" τον αλγόριθμο που επιθυμούμε και εκτελεί το clustering στα δεδομένα (rawData) Το αποτέλεσμα είναι ένας "πίνακας" που σε κάθε γραμμή έχει τα ζεύγη που συγχωνεύονται (είτε είναι συστάδες είτε σημεία στο πρώτο στάδιο), την ανομοιότητα και στην τελευταία στήλη του το βάθος του δέντρου. (μετρώντας από κάτω προς τα πάνω)

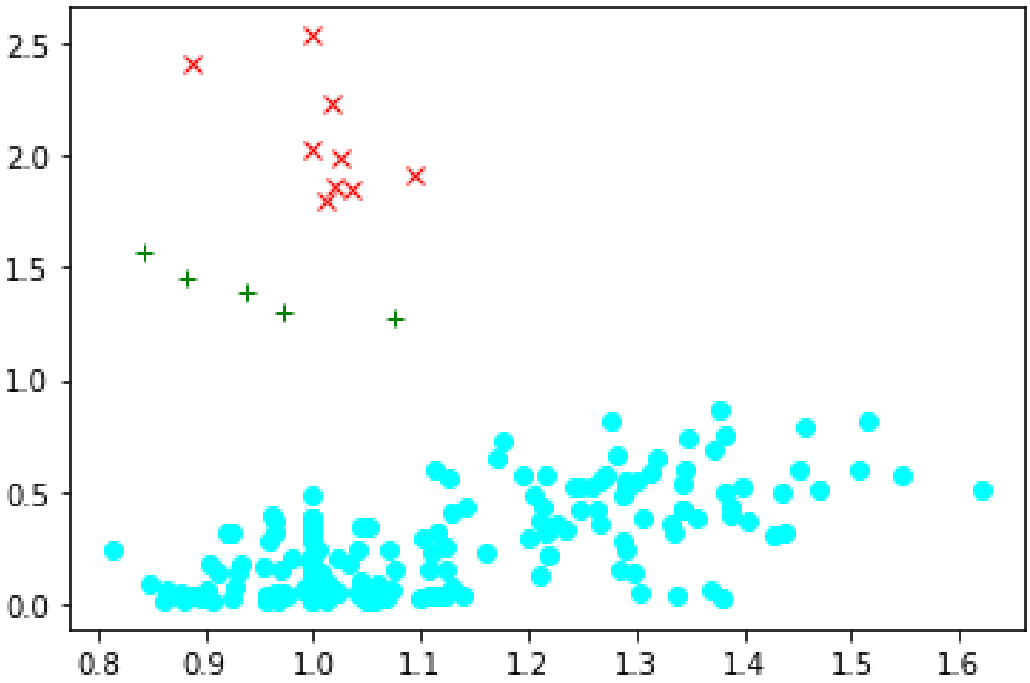
```
Z = linkage(rawData, 'centroid', optimal_ordering=True)
```

```
[[8.40000000e+01 1.20000000e+02 0.00000000e+00 2.00000000e+00]
[2.40000000e+01 1.05000000e+02 0.00000000e+00 2.00000000e+00]
[1.02000000e+02 1.32000000e+02 0.00000000e+00 2.00000000e+00]
[1.44000000e+02 1.20000000e+01 0.00000000e+00 2.00000000e+00]
[3.90000000e+01 6.00000000e+00 0.00000000e+00 2.00000000e+00]
[1.74000000e+02 5.70000000e+01 0.00000000e+00 2.00000000e+00]
[3.00000000e+01 4.80000000e+01 0.00000000e+00 2.00000000e+00]
[7.50000000e+01 1.23000000e+02 1.50000000e-05 2.00000000e+00]
[2.51000000e+02 3.30000000e+01 1.80000000e-05 3.00000000e+00] . . .
```

3.5 Εξαγωγή clusters

Για να εξάγουμε τα clusters χρησιμοποιούμε την συνάρτηση fcluster με το κριτήριο τερματισμού που στην περίπτωσή μας είναι ο αριθμός clusters που θέλουμε να μας επιστρέψει και που έχουμε ορίσει στο δεύτερο όρισμα.

```
k=3
clusters = fcluster(Z, k, criterion='maxclust')

mapping= { 1: ("cyan", "o"), 2: ("green", "+"), 3: ("red", "x"), 4: ("magenta", "x"),5: ("yellow", "x")}

for i,element in enumerate(rawData):
  plt.plot(element[0],element[1],c=mapping[clusters[i]][0], marker=mapping[clusters[i]][1])
plt.show()
```



Όπως παρατηρούμε αυτή η τεχνική clustering έχει σαν αποτέλεσμα την διαμόρφωση τριών clusters διαφορετικών από τις κλάσεις που παρατηρήσαμε στα πραγματικά δεδομένα. Μάλιστα το ένα cluster των πραγματικών δεδομένων ανήκει και στα 3 διαφορετικά clusters που δημιουργήσαμε με την παραπάνω μέθοδο.

Αυτό συμβαίνει διότι το κριτήριο σχηματισμού των clusters είναι η ευκλείδεια απόσταση, οπότε ανεξάρτητα από τον αλγόριθμο linkage που μπορεί να επιλέξει κανείς στην μέθοδο linkage είναι αδύνατο να εξαχθούν σωστά αποτελέσματα για τα δεδομένα εισόδου.

Οι μέθοδοι :

- centroid

Η μέθοδος ward χωρίζει τα δεδομένα όπως φαίνεται στο διάγραμμα που ακολουθεί:

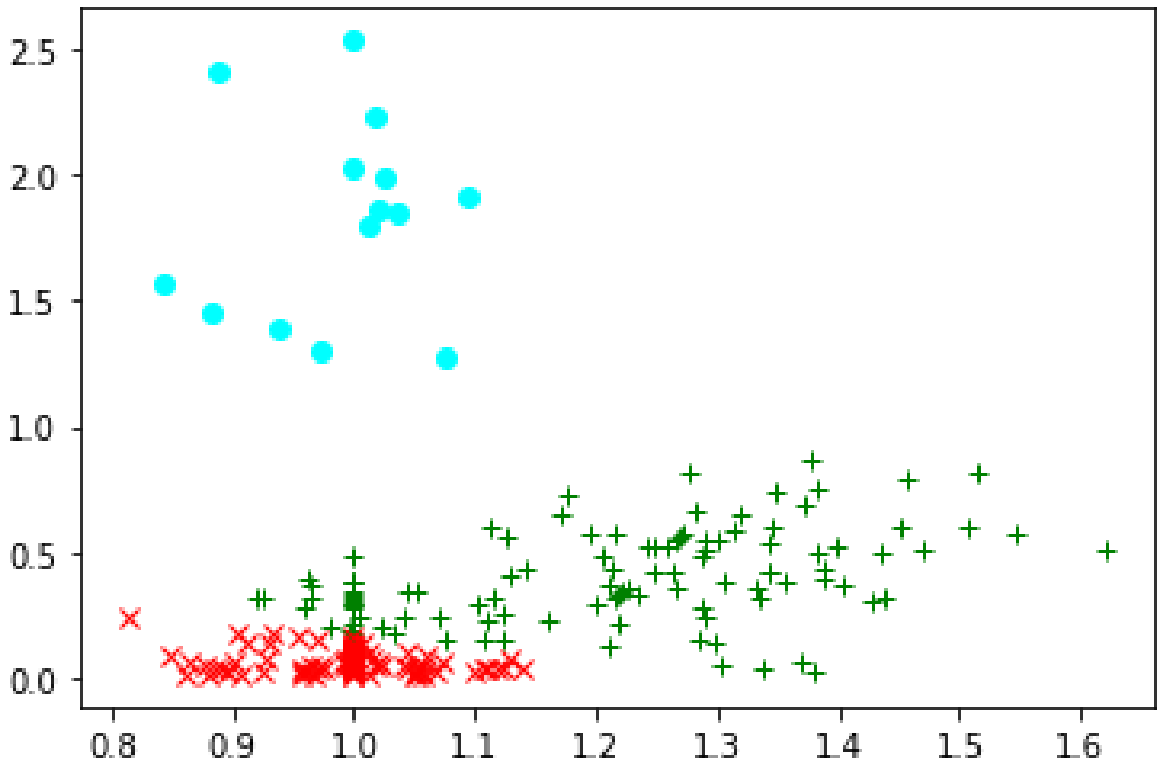
Η μέθοδος αυτή χρησιμοποιεί τον variance minimization algorithm του Ward, όπου κάθε καινούρια εισαγωγή υπολογίζεται από τον τύπο:

$$d(u,v) = \sqrt{\frac{|v|+|s|}{T}d(v,s)^2 + \frac{|v|+|t|}{T}d(v,t)^2 - \frac{|v|}{T}d(s,t)^2}$$

όπου το u είναι ένα μόλις συγχωνευμένο cluster και s,t,v είναι αχρησιμοποίητα clusters στο δάσος των κόμβων.

Και με αυτή τη μέθοδο υπάρχει αποτυχία στην διαδικασία αν και το αποτέλεσμα είναι πιο κοντά στην πραγματικότητα - έχει μικρότερο σφάλμα.

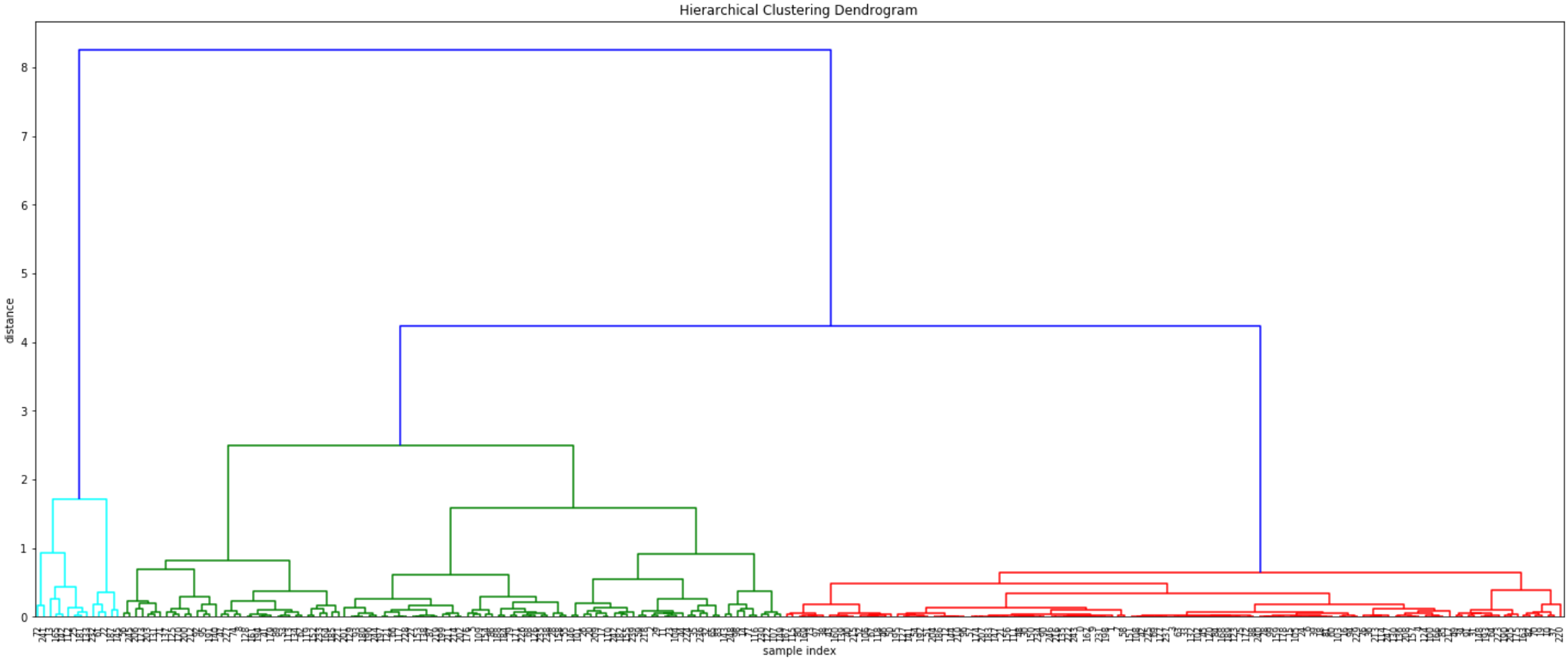
```
Z = linkage(rawData, 'ward', optimal_ordering=True)
k=3
clustersWard = fcluster(Z, k, criterion='maxclust')
mapping= { 1: ("cyan", "o"), 2: ("green", "+"), 3: ("red", "x"), 4: ("magenta", "x"),5: ("yellow", "x")}

for i,element in enumerate(rawData):
  plt.plot(element[0],element[1],c=mapping[clustersWard[i]][0],
marker=mapping[clustersWard[i]][1])
plt.show()
```



3.6 Δενδρόγραμμα

Εδώ κατασκευάζουμε και απεικονίζουμε το δενδρόγραμμα που παρουσιάζει όλη την ιεραρχία της από κάτω προς τα πάνω συσταδοποίησης.

```
rootnode, nodelist = hierarchy.to_tree(Z, rd=True)
L, M = leaders(Z, clustersWard)
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
  Z,
  leaf_rotation=90.,  # rotates the x axis labels
  leaf_font_size=8.,  # font size for the x axis labels
  #color_threshold=3.,
  link_color_func=lambda linkNode:
mapping[clustersWard[getLeafIDFromTreeNode(rootnode,nodelist,linkNode)]][0] if not linkNode
> L[0] else 'blue',


)
plt.figure(figsize=(10, 8))

plt.show()
```



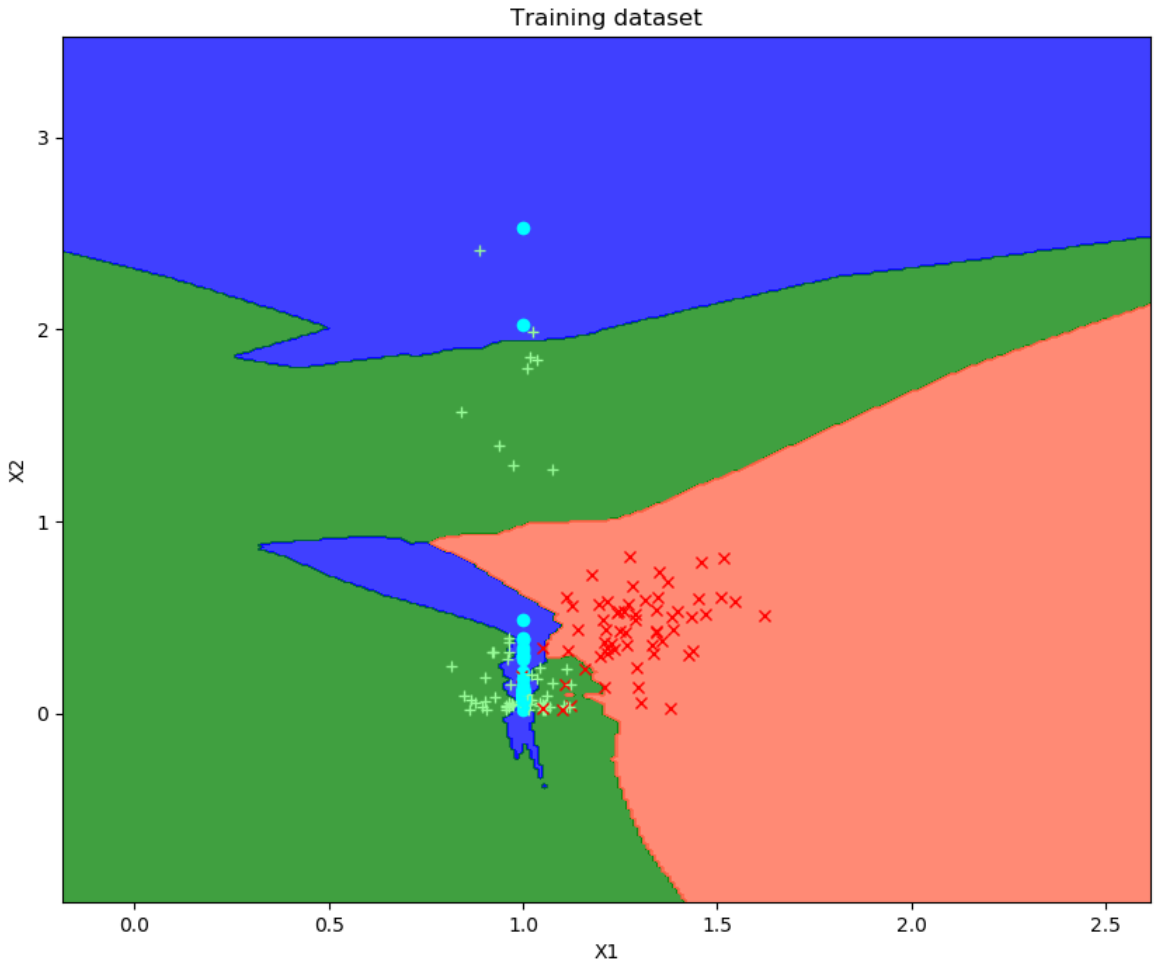
Το ύψος των κάθετων γραμμών του δενδρογράμματος υποδηλώνει την απόσταση μεταξύ των clusters που ενώθηκαν σε κάθε βήμα και ουσιαστικά καθορίζει και την σύσταση των clusters κατά την επιλογή του πλήθους των clusters που θέλουμε σαν έξοδο.

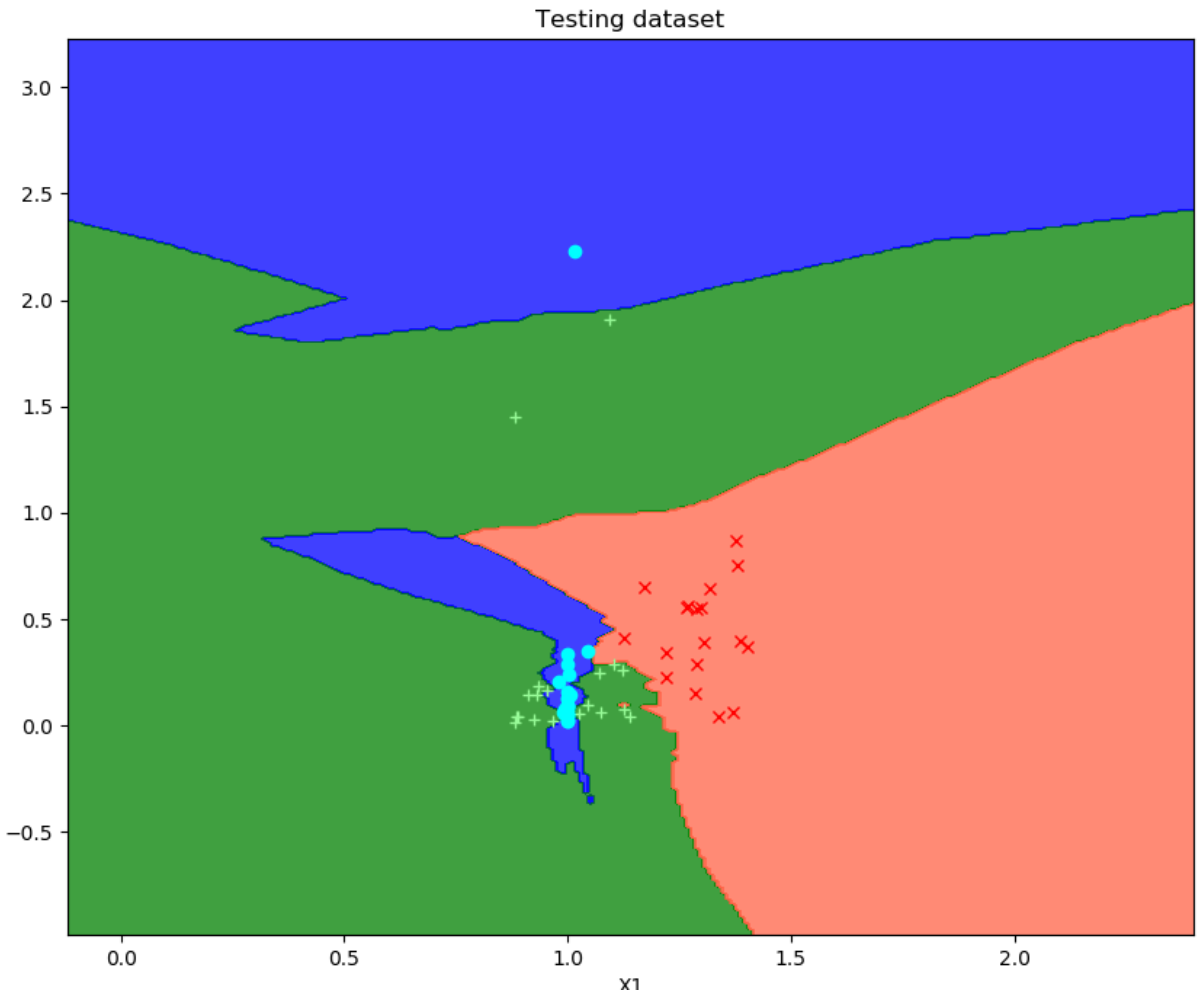
Οι κόμβοι έχουν τοποθετηθεί σε σειρά ανάλογα με το cluster που ανήκουν. Στον οριζόντιο άξονα βρίσκονται τα indices των κόμβων φύλλα του δέντρου.

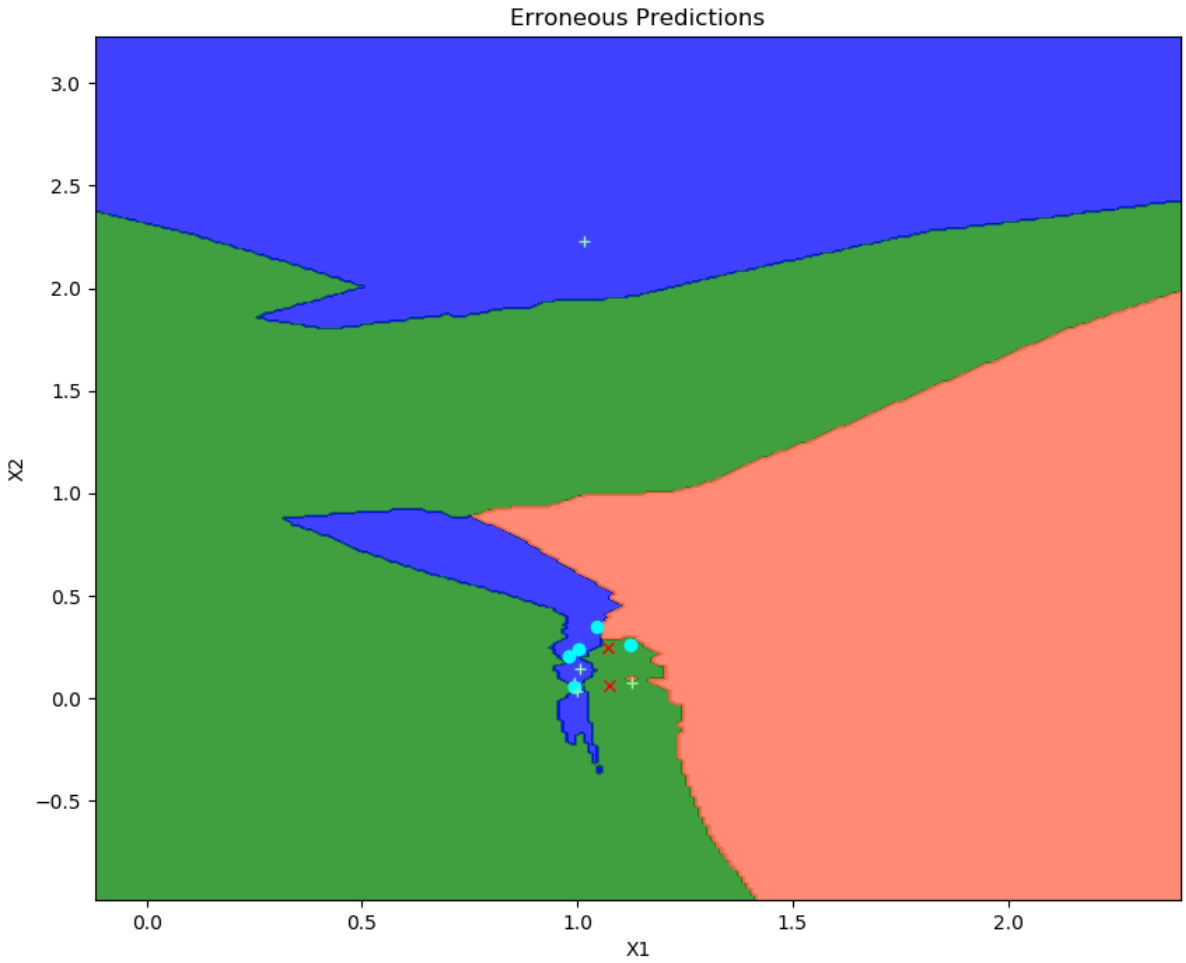
3.7 Εφαρμογή K-Nearest-Neighbors

Έχοντας υπόψη μας τις δυσκολίες λόγω της φύσης των δεδομένων και προσπαθώντας να βελτιώσουμε τα αποτελέσματα αποφασίσαμε να εφαρμόσουμε τον αλγόριθμο KNN K-Nearest-Neighbors ώστε να εξετάσουμε τα αποτελέσματα που θα μας έδινε δίνοντάς του ως σύνολο εκπαίδευσης ένα μέρος των δεδομένων.

Σε αυτό το κεφάλαιο συγκρίνονται οι δύο μέθοδοι. Εδώ θα πρέπει να σημειώσουμε ότι σε πολλές περιπτώσεις συσταδοποίησης δεδομένων δεν είναι δυνατό να έχει κανείς δεδομένα με ετικέτες και άρα δεν είναι δυνατό να εφαρμοστεί αυτός ο αλγόριθμος. Τέλος ο KNN δεν ανήκει στην ομάδα των hierarchical clustering αλγορίθμων.




Testing dataset
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
x1



Παρατηρούμε ότι οι λάθος προβλέψεις είναι λίγες, το ποσοστό τους όμως είναι τυχαίο και θα μπορούσε να είναι μεγαλύτερο ανάλογα με την θέση των δεδομένων προς ταξινόμηση. Η προβληματική συμπεριφορά είναι εμφανής και από το χρωματισμό των περιοχών απόφασης – η μπλε περιοχή θα έπρεπε να είναι μια ευθεία με x=1.

3.8 Κώδικας Hierarchical Clustering:

```
import pandas as pd
import xlsxwriter
import matplotlib.pyplot as plt
import numpy

from scipy.cluster.hierarchy import dendrogram, linkage,leaders
from scipy.cluster import hierarchy
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import inconsistent


def binSearchClusterNode(node, nodeID):
    #print(node.get_id(),nodeID)
    found = False
    if node.is_leaf() or node.get_id() < nodeID:
        return None,False

    if nodeID == node.get_id():
       # print("Found it")
        return node,True

    nodeL=node.get_left()
    nodeR=node.get_right()

    if nodeL.get_id() == nodeID:
        node,found = binSearchClusterNode(nodeL,nodeID)
    elif nodeR.get_id() == nodeID:
        node,found = binSearchClusterNode(nodeR, nodeID)
    else:
        node,found = binSearchClusterNode(nodeL,nodeID)
        if found == False:
            node,found = binSearchClusterNode(nodeR, nodeID)

    return node,found


def getLeafIDFromTreeNode(rootnode,nodelist,rootID):
   # print("BFS--")
    node,found=binSearchClusterNode(rootnode,rootID)
   # print(node.get_id())
    while not node.is_leaf():
        node = node.get_left()

   # print(node.get_id())

    return node.get_id()

pd.options.mode.chained_assignment = None  # default='warn'
df=pd.read_excel("features.xls",header=1,usecols=range(1,84),skiprows="1").dropna()

df=df.reset_index(drop=True)
print(df)

feature1=df.loc[0:2]
feature1.rename(index={0:'Ουδέτερη',1:'Χαρά',2:"Έκπληξη"}, inplace=True)
feature2=df.loc[3:5]
feature2=feature2.reset_index(drop=True)
feature2.rename(index={0:'Ουδέτερη',1:'Χαρά',2:"Έκπληξη"}, inplace=True)

print("Feature1:",feature1)
print("Feature2:",feature2)
```

```
data=[]
rawData=[]
for personID, col in feature1.iteritems():
    #print("person: ",personID)
    for cluster, element in enumerate(col):
        data.append([element, feature2[personID][cluster],cluster,personID])
        rawData.append([element, feature2[personID][cluster]])

data=numpy.array(data,dtype = object)
rawData=numpy.array(rawData)

mapping= { 0: ("cyan", "o"), 1: ("red", "x"), 2: ("green", "+") }

plt.figure(figsize=(10, 8))
for element in data:
    plt.plot(element[0],element[1],c=mapping[element[2]][0],
marker=mapping[element[2]][1])

plt.figure(figsize=(10, 8))
Z = linkage(rawData, 'centroid', optimal_ordering=True)

k=3
clusters = fcluster(Z, k, criterion='maxclust')

mapping= { 1: ("cyan", "o"), 2: ("green", "+"), 3: ("red", "x"), 4: ("magenta",
"x"),5: ("yellow", "x")}

for i,element in enumerate(rawData):
    plt.plot(element[0],element[1],c=mapping[clusters[i]][0],
marker=mapping[clusters[i]][1])

plt.figure(figsize=(10, 8))
Z = linkage(rawData, 'ward', optimal_ordering=True)
k=3
clustersWard = fcluster(Z, k, criterion='maxclust')
mapping= { 1: ("cyan", "o"), 2: ("green", "+"), 3: ("red", "x"), 4: ("magenta",
"x"),5: ("yellow", "x")}

for i,element in enumerate(rawData):
    plt.plot(element[0],element[1],c=mapping[clustersWard[i]][0],
marker=mapping[clustersWard[i]][1])


rootnode, nodelist = hierarchy.to_tree(Z, rd=True)
L, M = leaders(Z, clustersWard)
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
    #color_threshold=3.,
    link_color_func=lambda linkNode:
mapping[clustersWard[getLeafIDFromTreeNode(rootnode,nodelist,linkNode)]][0] if not
linkNode > L[0] else 'blue',


)


plt.show(
```

3.9 Κώδικας KNN:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier


df=pd.read_excel("features.xls",header=1,usecols=range(1,84),skiprows="1").dropna()

df=df.reset_index(drop=True)
print(df)

feature1=df.loc[0:2]
feature1.rename(index={0:'Ουδέτερη',1:'Χαρά',2:"Έκπληξη"}, inplace=True)
feature2=df.loc[3:5]
feature2=feature2.reset_index(drop=True)
feature2.rename(index={0:'Ουδέτερη',1:'Χαρά',2:"Έκπληξη"}, inplace=True)


print(feature1)
print(feature2)

data=[]
rawData=list()
fullRawData=list()
mapping= { 0:'Ουδέτερη',1:'Χαρά',2:"Έκπληξη" }

for personID, col in feature1.iteritems():
    #print("person: ",personID)
    for cluster, element in enumerate(col):
        data.append([element, feature2[personID][cluster],cluster,personID])
        #fullRawData.append([element, feature2[personID][cluster],mapping[cluster%3]])
        fullRawData.append([element, feature2[personID][cluster], cluster % 3])
        rawData.append([element, feature2[personID][cluster]])
        #print(element,feature2[personID][cluster])

#print(data)
data=np.array(data,dtype = object)
rawData=np.array(rawData)
print(rawData)
y = [row[2] for row in fullRawData]
print(y)
X=rawData
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
print (y)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
classifier = KNeighborsClassifier(n_neighbors = 2, metric = 'minkowski', p = 3)
classifier.fit(X_train, y_train)
# Predicting the Test set results
y_pred = classifier.predict(X_test)


# Making the Confusion Matrix
plt.figure(figsize=(10, 8))
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d")
```

```
mapping= { 0: ("cyan", "o"), 1: ("palegreen", "+"), 2: ("red", "x") }

# Visualising the Training set results
plt.figure(figsize=(10, 8))
from matplotlib.colors import ListedColormap

X_grid, y_grid = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_grid[:, 0].min() - 1, stop=X_grid[:, 0].max() +
1, step=0.01),
                     np.arange(start=X_grid[:, 1].min() - 1, stop=X_grid[:, 1].max() +
1, step=0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha=0.75,
             cmap=ListedColormap(('blue', 'green', 'tomato')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i,element in enumerate(X_train):
    print(element,y[i],i,mapping[y[i]][0])
    plt.plot(element[0],element[1],c=mapping[y_train[i]][0],
marker=mapping[y_train[i]][1])



plt.title('Training dataset')
plt.xlabel('X1')
plt.ylabel('X2')



print("predictions")
# Visualising the Predictions set results
from matplotlib.colors import ListedColormap
plt.figure(figsize=(10, 8))
X_grid, y_grid = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_grid[:, 0].min() - 1, stop=X_grid[:, 0].max() +
1, step=0.01),
                     np.arange(start=X_grid[:, 1].min() - 1, stop=X_grid[:, 1].max() +
1, step=0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha=0.75,
             cmap=ListedColormap(('blue', 'green', 'tomato')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i,element in enumerate(X_test):
    print(element,y[i],i,mapping[y[i]][0])
    plt.plot(element[0],element[1],c=mapping[y_pred[i]][0],
marker=mapping[y_pred[i]][1])

plt.title('Testing dataset')
plt.xlabel('X1')
plt.ylabel('X2')

# Visualising the Testing set actual clusters
from matplotlib.colors import ListedColormap
plt.figure(figsize=(10, 8))
X_grid, y_grid = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_grid[:, 0].min() - 1, stop=X_grid[:, 0].max() +
1, step=0.01),
                     np.arange(start=X_grid[:, 1].min() - 1, stop=X_grid[:, 1].max() +
1, step=0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha=0.75,
```

```
            cmap=ListedColormap(('blue', 'green', 'tomato')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i,element in enumerate(X_test):
    plt.plot(element[0],element[1],c=mapping[y_test[i]][0],
marker=mapping[y_test[i]][1])

plt.title('Testing dataset actual clusters')
plt.xlabel('X1')
plt.ylabel('X2')

# Visualising the erroneous predictions
from matplotlib.colors import ListedColormap
plt.figure(figsize=(10, 8))
X_grid, y_grid = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_grid[:, 0].min() - 1, stop=X_grid[:, 0].max() +
1, step=0.01),
                     np.arange(start=X_grid[:, 1].min() - 1, stop=X_grid[:, 1].max() +
1, step=0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha=0.75,
             cmap=ListedColormap(('blue', 'green', 'tomato')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i,element in enumerate(X_test):
    if(y_test[i] != y_pred[i]):
        plt.plot(element[0],element[1],c=mapping[y_pred[i]][0],
marker=mapping[y_pred[i]][1])
        print(element[0],element[1],mapping[y_pred[i]][0],y_train[i])

plt.title('Predicted cluster - Errors in prediction')
plt.xlabel('X1')
plt.ylabel('X2')

print("---")
# Visualising the erroneous predictions
from matplotlib.colors import ListedColormap
plt.figure(figsize=(10, 8))
X_grid, y_grid = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_grid[:, 0].min() - 1, stop=X_grid[:, 0].max() +
1, step=0.01),
                     np.arange(start=X_grid[:, 1].min() - 1, stop=X_grid[:, 1].max() +
1, step=0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha=0.75,
             cmap=ListedColormap(('blue', 'green', 'tomato')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i,element in enumerate(X_test):
    if(y_test[i] != y_pred[i]):
        plt.plot(element[0],element[1],c=mapping[y_train[i]][0],
marker=mapping[y_train[i]][1])
        print(element[0],element[1],mapping[y_train[i]][0],y_train[i])

plt.title('Erroneous Predictions')
plt.xlabel('X1')
plt.ylabel('X2')

plt.show()
```

4. Θέμα 4ο - Genetic Travelling Salesman Problem

4.1 Εισαγωγή

4.1.1 Βασικές Πληροφορίες Γενετικών Αλγορίθμων

Οι **Γενετικοί αλγόριθμοι** ανήκουν στο κλάδο της επιστήμης υπολογιστών και αποτελούν μια μέθοδο αναζήτησης βέλτιστων λύσεων σε συστήματα που μπορούν να περιγραφούν ως μαθηματικό πρόβλημα. Είναι χρήσιμοι σε προβλήματα που περιέχουν πολλές παραμέτρους /διαστάσεις και δεν υπάρχει αναλυτική μέθοδος που να μπορεί να βρει το βέλτιστο συνδυασμό τιμών για τις μεταβλητές ώστε το υπό εξέταση σύστημα να αντιδρά με όσο το δυνατόν με το επιθυμητό τρόπο.

Ο τρόπος λειτουργίας των Γενετικών Αλγορίθμων είναι εμπνευσμένος από τη βιολογία. Χρησιμοποιεί την ιδέα της εξέλιξης μέσω γενετικής μετάλλαξης, φυσικής επιλογής και διασταύρωσης. Οι Γενετικοί Αλγόριθμοι είναι αρκετά απλοί στην υλοποίησή τους. Οι τιμές για τις παραμέτρους του συστήματος πρέπει να κωδικοποιούνται με τρόπο ώστε να αναπαρασταθούν από μια μεταβλητή που περιέχει σειρά χαρακτήρων ή δυαδικών ψηφίων (0/1). Αυτή η μεταβλητή μιμείται το γενετικό κώδικα που υπάρχει στους ζωντανούς οργανισμούς. Αρχικά, ο Γενετικός Αλγόριθμος παράγει πολλαπλά αντίγραφα της μεταβλητής/γεννητικού κώδικα, συνήθως με τυχαίες τιμές, δημιουργώντας ένα πληθυσμό λύσεων. Κάθε λύση (τιμές για τις παραμέτρους του συστήματος) δοκιμάζεται για το πόσο κοντά φέρνει την αντίδραση του συστήματος στην επιθυμητή, μέσω μιας συνάρτησης που δίνει το μέτρο ικανότητας της λύσης και η οποία ονομάζεται συνάρτηση ικανότητας (Σ.Ι).

Οι λύσεις που βρίσκονται πιο κοντά στην επιθυμητή, σε σχέση με τις άλλες, σύμφωνα με το μέτρο που μας δίνει η Σ.Ι, αναπαράγονται στην επόμενη γενιά λύσεων και λαμβάνουν μια τυχαία μετάλλαξη. Επαναλαμβάνοντας αυτή τη διαδικασία για αρκετές γενιές, οι τυχαίες μεταλλάξεις σε συνδυασμό με την επιβίωση και αναπαραγωγή των γονιδίων/λύσεων που πλησιάζουν καλύτερα το επιθυμητό αποτέλεσμα θα παράγουν ένα γονίδιο/λύση που θα περιέχει τις τιμές για τις παραμέτρους που ικανοποιούν όσο καλύτερα γίνεται την Σ.Ι.

Υπάρχουν διάφορες εκδοχές της παραπάνω διαδικασίας για τους Γ.Α από τις οποίες κάποιες περιλαμβάνουν και τη διασταύρωση (ζευγάρωμα) γονιδίων/λύσεων ώστε ο αλγόριθμος να φτάσει στο αποτέλεσμα πιο γρήγορα. Καθώς υπάρχει το στοχαστικό (τυχαίο) συστατικό της μετάλλαξης και ζευγαρώματος, κάθε εκτέλεση του Γ.Α μπορεί να συγκλίνει σε διαφορετική λύση και σε διαφορετικό χρόνο. Η απόδοση του Γ.Α εξαρτάται επί το πλείστον από την συνάρτηση ικανότητας και συγκεκριμένα από το κατά πόσο το μέτρο της περιγράφει την βέλτιστη λύση. Οι γενετικοί αλγόριθμοι είναι ένα πεπερασμένο σύνολο οδηγιών για την εκπλήρωση ενός έργου, το οποίο δεδομένης μιας αρχικής κατάστασης θα οδηγήσει σε μια αναγνωρίσιμη τελική κατάσταση, και το οποίο προσπαθεί να μιμηθεί την διαδικασία της βιολογικής εξέλιξης. Οι γενετικοί αλγόριθμοι προσπαθούν να βρουν τη λύση ενός προβλήματος με το να προσομοιώνουν την εξέλιξη ενός πληθυσμού «λύσεων» του προβλήματος.

4.1.2 Σκοπός της εργασίας

Στη παρούσα εργασία, σκοπός είναι η επίλυση του προβλήματος του περιοδεύοντας πωλητή, ώστε να γίνει κατανοητή η λειτουργία του και να δοκιμαστεί με διάφορα πλήθη πόλεων και γενεών. Γλώσσα υλοποίησης είναι η Python.

4.2 Περιγραφή Προβλήματος

Το προβλήμα του περιοδεύοντας πωλητή προσπαθεί να απαντήσει το εξής ερώτημα: «Δοθέντος μιας λίστας από πόλεις και των αποστάσεων μεταξύ τους , ποια είναι η μικρότερη διαδρομή ώστε να περάσουμε από κάθε πόλη ακριβώς μια φορά;». Ο γενετικός αλγόριθμος με τον οποίο επιλύουμε το παραπάνω πρόβλημα περιλαμβάνει τα εξής βήματα :

α) Δημιουργία **Πληθυσμού** (Population)

β) Υπολογισμός της τιμή **fitness** για κάθε πιθανή λύση του προβλήματος.

γ) **Επιλογή** Γονέων για την επόμενη γενεά(Parent Selection)

δ) **Διασταύρωση** (Crossover)

ε) **Μετάλλαξη**(Mutation)

στ) **Elitism**

4.3 Δημιουργία Πληθυσμού (Population)

Για την δημιουργία πληθυσμού δημιουργούμε τυχαία λίστες απο διαδρομές πόλεων. Αυτό επιτυγχάνεται με την συνάρτηση <<shuffle>> που μας δίνει η python.

```
def generatePop(pop, popSize, cityOrder):
for i in range(popSize):
                  temp = cityOrder.copy()
            random.shuffle(temp)
            pop.append(temp)
return pop
```

4.4 Υπολογισμός Fitness

Για να υπολογίσουμε το fitness κάθε πιθανής λύσης , βρισκουμε το άθροισμα των αποστάσεων από όλες τις πόλεις στην συγκεκριμένη διαδρομή και το διαιρούμε με το συνολικό fitness της γεναιάς.

Κάθε διαδρομή από πόλεις (route) όσο πιο μεγάλο fitness έχει τόσο μεγαλύτερη πιθανότητα να περάσει στην επόμενη γεναιά.

```
def fitness(dist, cities, pop, fit):
    for i in range(len(pop)):
        sum = 0
        #pop[i].append(pop[i][0])
        for j in range(len(pop[0])-1):
            c1 = pop[i][j]
            c2 = pop[i][j+1]
            sum += dist[c1][c2]
        fC = pop[i][0] #add distance to return to origin city
        lC = pop[i][-1]
        sum += dist[lC][fC]
        fit.append(sum)
```

4.5 Επιλογή Γονέων (Parent Selection)

Η επιλογή των γονέων γίνεται με Fitness Proportionate Selection. Δηλαδή κάθε διαδρομή απο πόλεις που έχει σχετικά μεγάλο fitness έχει περισσότερες πθανότητες να συνεχίσει στην επόμενη γενεά. Επιπροσθετα χρησιμοποιείται η μέθοδος Roulette Wheel Selection για την επιλογή των γονέων. Καθε διαδρομή απο πόλεις (individual) έχοντας μια fitness πιθανότητα επιλογής αποτελέι κομμάι μιας νοητής ρουλέττας. Ετσ , παράγουμε 2 τυχαίους αριθμού και επιλέγουμε τελικά όλες τους γονείς (διαδρομές) που βρίσκονται ανάμεσα σε αυτές.

```
def selection(popFit, popSize, numElite):
    totalFit = 0
    for i in range(len(popFit)):    #Find Iverse proportional slices
        fitVal = 1/popFit[i][1]
        newTup = (popFit[i][0], fitVal)
        popFit[i] = newTup
    totalFit = 0
    for i in range(len(popFit)):    #Final Total Fitness Value or normalized
values
        totalFit += popFit[i][1]

    for i in range(len(popFit)):
        fitVal = popFit[i][1]
        newTup = (popFit[i][0], (fitVal/totalFit))
        popFit[i] = newTup

    popFit = sorted(popFit, key = lambda x : x[1])  #Sort Population by
Fitness

    accNorm = []
    normSum = 0
    for i in range(len(popFit)):    #Create list of the accumulate
normalization values
```

```
        accNorm.append(normSum + popFit[i][1])
        normSum += popFit[i][1]

    parents = []
    indexArr = []
    intPop = int(popSize)
    for i in range(int(popSize-numElite/2)):
        parentss = []
        for j in range(2):
            randNum = random.randint(0,100)/100
            for j in range(len(accNorm)):
                if (j == 0):
                    if (randNum < accNorm[j]):
                        parentss.append(popFit[j][0])
                else:
                    if (randNum < accNorm[j] and randNum > accNorm[j-1]):
                        parentss.append(popFit[j][0])
        parents.append(parentss)

    #Ensure every value in parents is a couple and not a single
    for i in range(len(parents)-1):
        if (len(parents[i]) == 0):
            parents.pop(i)
        if (len(parents[i]) != 2):
            parents[i].append(parents[i][0])

    return parents
```

4.6 Διασταύρωση (Crossover)

Η διασταύρωση (crossover) 2 γονέων υλοποιήθηκε με One Point Crossover. Εχοντας επιλέξει 2 γονείς και έναν τυχαίο αριθμό K για crossover δημιουργούμε το 1 παιδί από τον πρώτο πατέρα συμπληρώνοντας τα K στοιχειά από την αρχή του 2ου γονέα.Το 2ο παιδί δημιουργείται από τον 2ο γονέα συμπληρώντας τα τελευταία K στοιχεία του 1ου γονέα. Τα διπλότυπα στοιχεία παραλείπονται ώστε να μην υπαρχεί σε κάθε διαδρομή δύο φορές η ίδια πόλη.

```
def crossover(parents, popSize, pCross, numElite):
    newPop = []
    for i in range(int(len(parents)-numElite)):
        prob = random.randint(0,100)/100
        if (prob > pCross): #Dont Crossover
            if (len(parents[i]) == 1):
                parents[i].append(parents[i][0])
            newPop.append(parents[i][0])
            newPop.append(parents[i][1])
            pass
        elif (prob <= pCross):  #Perform Crossover
            portionArr = []
            child1 = []
```

```
            child2 = []
            childArr = [child1, child2]
            cut = []
            cut2 = []
            cutArr = [cut, cut2]

            acc = 0

            if (len(parents[i]) == 1):
                parents[i].append(parents[i][0])

            #print(len(parents[i]))
            parentsArr = [parents[i][0],parents[i][1]]   #Parents
            for j in range(2):  #One Iteration per parent
                portionInd = random.sample(range(0, len(parents[j][0])+1),
2)  #Indexes used for isolation crossover portion
                portionInd = sorted(portionInd)
                portion = parentsArr[j][portionInd[0]:portionInd[1]]
                portionArr.append(portion)

                for k in range(len(parentsArr[j])):     #Check if every
value from pX is in the portion, if it is:ignore, if not:append to cutArr
                    if (j == 0):
                        if (parentsArr[1][k] not in portion):
                            cutArr[0].append(parentsArr[1][k])
                    if (j == 1):
                        if (parentsArr[0][k] not in portion):
                            cutArr[1].append(parentsArr[0][k])

                tempInd = 0
                tempInd2 = 0

                for l in range(len(parentsArr[j])):
                    if (l < portionInd[0]):
                        childArr[j].append(cutArr[j][l])
                        tempInd +=1
                    if (l >= portionInd[0] and l < portionInd[1]):
                        childArr[j].append(portionArr[j][tempInd2])
                        tempInd2 +=1
                    if (l >= portionInd[1]):
                        childArr[j].append(cutArr[j][tempInd])
                        tempInd +=1
            newPop.append(childArr[0])
            newPop.append(childArr[1])

    return newPop
```

4.7 Μετάλλαξη (Mutation)

Η μετάλλαξη υλοποιήθηκε και δοκιμάστηκε με 3 διαφορετικούς Τρόπους:

A) Insert Mutation

```
def insMut(pop, pMut, numElite): #Takes a child after crossover
    for i in range(int(len(pop)-numElite)):
        prob = random.randint(0,100)/100
        if (prob <= pMut):  #Mutate
            chrom = pop[i]
            ind = sorted(random.sample(range(0,len(pop[i])), 2))

            val1 = pop[i][ind[0]]
            val2 = pop[i][ind[1]]
            indAcc = val2
            while (pop[i][ind[0]+1] != val2): #While the number after the
first index isnt the desired value
                temp = pop[i][ind[1]-1]
                pop[i][ind[1]-1] = pop[i][ind[1]]
                pop[i][ind[1]] = temp
                ind[1] -= 1

    return pop
```

B)Inverse Mutation

```
def invMut(pop, pMut, numElite):
    for i in range(int(len(pop)-numElite)):
        prob = random.randint(0,100)/100
        if (prob <= pMut):  #Mutate
            #print("Mutate")
            chrom = pop[i]
            #print(pop[i])
            ind = sorted(random.sample(range(0,len(pop[i])), 2))

            s1 = chrom[0:ind[0]]
            seg = chrom[ind[0]:ind[1]]
            s2 = chrom[ind[1]:]

            seg.reverse()
            newChrom = s1 + seg + s2
            pop[i] = newChrom
            #print(pop[i])
    return pop
```

Γ)Swap Mutation

```
def swapMut(pop, pMut, numElite): #Takes a child after crossover
    for i in range(int(len(pop)-numElite)):
        prob = random.randint(0,100)/100
        if (prob <= pMut):  #Mutate
            chrom = pop[i]
            ind = sorted(random.sample(range(0,len(pop[i])), 2))


            val1 = pop[i][ind[0]]
            val2 = pop[i][ind[1]]
            temp = val1
            pop[i][ind[0]] = val2
            pop[i][ind[1]] = temp

    return pop
```

Με την χρήση Mutation δίνουμε τυχαίοτητα σε κάθε διαδρομή από πόλεις (individual) ώστε να μην ξανά επαναλαμβάνεται στην επόμενη γενεά και επομένως να βελτιώνεται το fitness.

4.8 Elitism

Χρησιμοποιώντας Elitism παιρνάμε ένα συγκεκριμένο ποσοστό από διαδρομές πόλεων(individual) με βάση το fitness στην επόμενη γενεά χωρίς να κάνουμε mutation.Παίζει σημαντικό ρόλο στην βελτίωση του fitness από γενεά σε γενέα .

4.9 Αποτελέσματα

Τα αποτελέσματα έπειτα από δοκιμές που πραγματοποιήθηκαν έδειξαν πώς ο γενετικός αλγόριθμος κατάφερε να βρεί optimal

Λύση για διάφορα TSPs datasets.Οι δοκιμές και τα στατιτικά που εξάγαμε επικεντρώθηκαν στην μεθόδους με τις οποίες κάνουμε Mutate και αυτό γιατί το crossover και το Parent Selection ήταν κοινό για όλες τις μετρήσεις.

Από τον Πίνακα 1 βλέπουμε πως όλες οι μέθοδοι που χρησιμοποιήσαμε για mutate λειτούργησαν αρκετά καλά.Συγκεκριμένα, η Invert και Insert Mutation βρήκαν την optimal διαδρομή 1127 με επιτυχία 100% και η swap mutation με 90% καθώς βρήκε την 1128 διαδρομή.

Από τον πίνακα 2 βλέπουμε ότι η Inverse mutation ήταν η πιο αποτελεσματική με επιτυχία 50% στην optimal διαδρομή.Η Insert και swap mutation είχαν 40% επιτυχία.

Από τον Πίνακα 3 παρατηρούμε ότι είναι πιο έντονες οι διαφορές των mutate μεθόδων.Συγκεκριμένα, καμία δεν μπόρεσε να βρεί την optimal διαδρομή 100% ενώ η Inverse και swap μέθοδος το έκανε με επιτυχία 20%.

Από τον Πίνακα 4 παρατηρούμε ότι καμία μέθοδος δεν βρήκε την optimal λύση. Η Inverse mutation ήταν η πιο σταθερή από όλες.Στην συνέχεια η Swap mutation και τέλος η Insert.

Table I (Opt. Dist. = 1127)

10 TSP	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Avg. Fit	Avg. Conv.	Avg. Impr.
Inv. Mut	1127	1127	1127	1127	1127	1127	1127	1127	1127	1127	1127	15.7	169%
Ins. Mut	1127	1127	1127	1127	1127	1127	1127	1127	1127	1127	1127	15.1	172%
Swap Mut.	1127	1127	1127	1127	1127	1127	1127	1127	1127	1128	1127.1	17.2	168%

Table II (Opt. Dist. = 2085)

17 TSP	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Avg. Fit	Avg. Conv	Avg. Impr.
Inv. Mut	2085	2085	2090	2085	2085	2090	2090	2085	2088	2090	2087.3	73.2	189%
Ins. Mut	2085	2090	2123	2085	2146	2085	2118	2090	2085	2152	2105.9	59.9	185%
Swap Mut.	2158	2085	2090	2085	2153	2116	2090	2090	2085	2085	2103.7	65.7	181%

Table III (Opt. Dist. = 937)

26 TSP	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Avg. Fit	Avg. Conv.	Avg. Impr.
Inv. Mut	953	937	962	940	972	956	953	955	1003	937	956.8	131.8	236%
Ins. Mut	1003	975	993	994	974	1013	989	978	974	995	988.8	133.8	237%
Swap Mut.	978	937	959	970	1001	955	1035	978	1003	937	975.3	144.3	228%

Table IV (Opt. Dist. = 10,628)

48 TSP	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Avg. Fit	Avg. Conv.	Avg. Impr.
Inv. Mut	34737	35626	35205	34544	35270	35471	35371	36934	35248	34257	35266.3	340.7	363%
Ins. Mut	35107	36218	41204	39237	40679	47543	43761	42398	35826	46024	40799.7	330.9	338%
Swap Mut.	39523	37282	36882	36018	40046	43114	42455	48760	34772	38024	39687.6	366.9	329%