ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΜΕΡΟΣ Β’ 2021-2022

ΣΠΕΝΤΖΑΡΗΣ ΠΑΝΑΓΙΩΤΗΣ 1071110

Υπολογιστική Νοημοσύνη

Κώδικας Link: https://github.com/Pspetz/Computational-Intelligence-partB

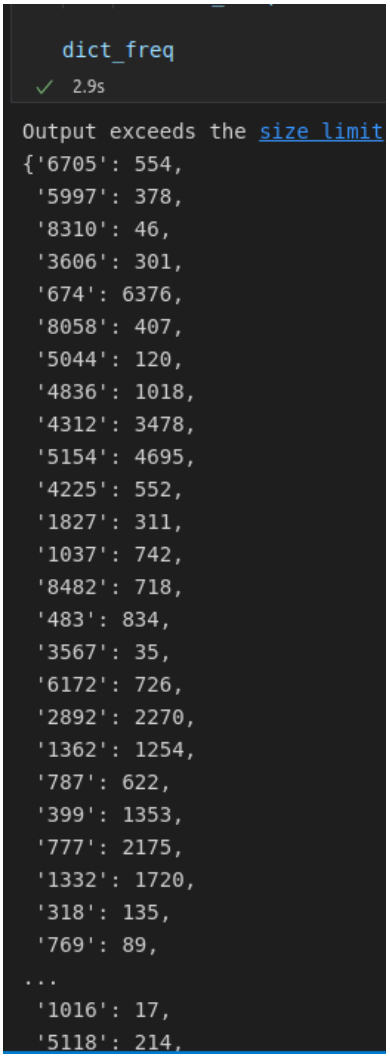
Α1. ΣΧΕΔΙΑΣΜΟΣ ΓΑ

Β. Εντοπισμός σχετικών λέξεων από ένα σώμα κειμένου με χρήση Γ

Ο στόχος είναι η δημιουργία ενός γενετικού αλγορίθμου που θα εντοπίζει ποιες λέξεις είναι σημαντικές σε ένα κείμενο και ποιες όχι. Για να γίνει αυτό πρέπει να προ επεξεργαστούμε τα δεδομένα με την μέθοδο tf-idf που θεωρείται η μετρική της σημαντικότητας μίας λέξης .Έτσι μπορούμε να χρησιμοποιήσουμε τον μέσο όρο εμφάνισης μιας λέξης από όλα τα κείμενα και να βρούμε την σημαντικότητα της. Πριν όμως εφαρμόσουμε την τεχνική tf-idf θα πρέπει να καθαρίσουμε τα δεδομένα έτσι ώστε να διαγραφούν οι αγκύλες <> οι οποίες δεν θεωρούνται λέξεις στο κείμενο .Παρατηρούμε ότι αρχικά με την χρήση regex αφαιρούμε τα <> και στην συνέχεια το κάνουμε append σε ένα άδειο πίνακα. Στην συνέχεια στο words περνιέται κάθε λέξη που εμφανίζεται μέσα στο κείμενο και στην συνέχεια με το tokenize βρίσκουμε ποιες λέξεις έχουμε στο κείμενο(8520).Τέλος με μία διπλή συνθήκη δημιουργούμε το dictionary dict_freq όπου αποθηκεύεται εκεί η λέξη και ο αριθμός εμφάνισης της λέξης μέσα σε όλα τα κείμενα.

```
#clear data
clear_file=[]
for i in range(len(file)):
    x=re.sub('<.*?>','',file[i])
    clear_file.append(x)

clear_file=clear_file[:]

#perasma tou clear keimenou sto words strin
words = ''
for line in clear_file:
    words += line


tokenized_words = word_tokenize(words) # li
WORD_LIST = list(set(tokenized_words)) # cr
dictionary_size = len(WORD_LIST) #6853


#Dhmiourgia dictionary lekseis kai suxnotit
for word in tokenized_words: #gia kathe wor
    if word not in dict_freq:
        dict_freq[word] = 1
    else:
        dict_freq[word] += 1
```

```
tokenized_words
✓ 2.9s

Output exceeds the size limit
['6705',
 '5997',
 '8310',
 '3606',
 '674',
 '8058',
 '5044',
 '4836',
 '4312',
 '5154',
 '8310',
 '4225',
 '1827',
 '1037',
 '8482',
 '483',
 '3567',
 '6172',
 '6172',
 '2892',
 '1362',
 '787',
 '399',
```



α) ΚΩΔΙΚΟΠΟΙΗΣΗ

Πρώτο στάδιο για την δημιουργία ενός Γενετικού αλγορίθμου είναι η Κωδικοποίηση. Οι γενετικοί αλγόριθμοι χρησιμοποιούν συνήθως δυαδική κωδικοποίηση, όπου οι είσοδοι και έξοδοι του προβλήματος αναπαριστάτε με συμβολοσειρές(strings) σταθερού μήκους ενός συγκεκριμένου αλφαβήτου .Μια συμβολοσειρά έχει ως ρόλο αντίστοιχο του χρωμοσώματος στον γενετικό. Επίσης κάθε θέση στην συμβολοσειρά έχει ρόλο αντίστοιχο με αυτό του γενετικού υλικού στους βιολογικούς μηχανισμούς δηλαδή γονίδιο μέσα σε χρωμόσωμα. Επιπρόσθετα η κωδικοποίηση σε 0 και 1 είναι η πιο εύκολη μορφή καθώς ορίζουμε τυχαία στοιχεία του χρωμοσώματος-ατόμου. Επίσης το πρόβλημα που καλούμαστε να λύσουμε είναι OneMax δηλαδή θέλουμε να δούμε ποιες λέξεις είναι σημαντικές στα κείμενα που έχουμε και ποιες όχι άρα αναπαριστάτε σε 0 και 1,δηλαδή τιμή 0 για την ασήμαντη λέξη ενώ 1 για την λέξη που θεωρείται σημαντική.

β) ΑΡΧΙΚΟΣ ΠΛΗΘΥΣΜΟΣ

Ο γενετικός αλγόριθμος δημιουργεί με τυχαίο τρόπο τον αρχικό πληθυσμό όπου κάθε άτομο θα παίρνει μία τιμή , 1 ή 0.

γ) ΔΙΑΔΙΚΑΣΙΑ ΕΠΙΔΙΟΡΘΩΣΗΣ

Οι τελεστές υποστηρίζουν ο ένας τον άλλον , αφού αν αφαιρεθεί η διασταύρωση το σύστημα χρησιμοποιεί την μετάλλαξη για την ανίχνευση μίας συγκεκριμένης περιοχής χωρίς την δυνατότητα σχεδιασμού καλύτερων ατόμων που οδηγεί αυτονόητα το σύστημα σε μειωμένη απόδοση. Ενώ αντίθετα περιμένουμε πως αν καταργηθεί η μετάλλαξη τότε ο αλγόριθμος που θα φτιάξουμε θα συγκλίνει γρήγορα ενώ θα λείπουν σημαντικά άτομα.

δ) ΥΠΟΛΟΓΙΣΜΟΣ TF-IDF

TF: Είναι το μέτρο της συχνότητας των λέξεων σε ένα έγγραφο. Είναι ο λόγος του αριθμού των φορών που εμφανίζεται η λέξη σε ένα έγγραφο σε σύγκριση με τον συνολικό αριθμό των λέξεων σε αυτό το έγγραφο.

```
tf(t,d) = count of t in d / number of words in d
```

IDF: Οι λέξεις που εμφανίζονται σπάνια στο σώμα έχουν υψηλή βαθμολογία. Είναι το αρχείο καταγραφής της αναλογίας του αριθμού των εγγράφων προς τον αριθμό των εγγράφων που περιέχουν τη λέξη.

```
idf(t) = log(N/(df + 1))
```

Για να υπολογίσουμε το tf-idf όπως προαναφέρθηκε πρέπει να κάνουμε καθαρισμό στα δεδομένα καθώς και να βρούμε το πόσες φορές εμφανίζεται κάθε λέξη στα κείμενα που έχουμε.

Στην συνέχεια δημιουργώ 2 συναρτήσεις όπου η πρώτη χρησιμοποιείται μόνο για τον υπολογισμό του TF, ενώ η δεύτερη για τον υπολογισμό του IDF.Η πρώτη συνάρτηση παίρνει σαν ορίσματα το dictionary με την συχνότητα εμφάνισης κάθε λέξεις ενώ σαν δεύτερο όρισμα τις λέξεις που έχουμε(8520). Ενώ ο υπολογισμός της συνάρτησης idf παίρνει σαν όρισμα μόνο το dictionary και αποθηκεύουμε πάλι τις τιμές σε ένα άδειο dictionary.

```python
#TF = (Frequency of the word in the sentence) / (Total number of words in the sentence)
def computeTF(wordDict,bow):
    tfDict={}
    bowCount=len(bow)
    for word,count in wordDict.items():
        tfDict[word] = count/float(bowCount)
    return tfDict

tf=computeTF(dict_freq,tokenized_words)

#IDF: Log((Total number of sentences (documents))/(Number of sentences (documents) containing the word))
def computeIDF(docList):
    idfDict = {}
    N = len(docList)

    idfDict = dict.fromkeys(dict_freq,0)
    for word, val in idfDict.items():
        idfDict[word] = math.log10(N / (float(val) + 1))

    return idfDict

idf=computeIDF(dict_freq)
len(idf)
```

```
8520
```

Τέλος η συνάρτηση που δημιούργησα συνδυάζει τις συναρτήσεις tf,idf ,ώστε να μας δώσουν την επιθυμητή έξοδο που θέλουμε .Σαν όρισμα παίρνει το tf,idf και η έξοδος αποθηκεύεται σε μία νέα μεταβλητή που περιέχει τελικά την το TF_IDF της κάθε λέξης.

```python
#FINAL TF-IDF
def computeDFIDF(tfbow,idfs):
    tfidf={}
    for word,val in tfbow.items():
        tfidf[word] = val*idfs[word]

    return tfidf
```

```python
#final TF-IDF
Tf_idf=computeDFIDF(tf,idf)
```

```
Tf_idf

✓ 0.1s

Output exceeds the size limit. Open
{'6705': 0.0021653505644432564,
 '5997': 0.0014774413590854785,
 '8310': 0.0001797944511992595 3,
 '3606': 0.0011764810828473289,
 '674': 0.024921074366227803,
 '8058': 0.00159079003561084,
 '5044': 0.0004690290031285032,
 '4836': 0.003978929376540135,
 '4312': 0.01359402394067445,
 '5154': 0.018350759747402686,
 '4225': 0.002157533414391115,
 '1827': 0.0012155668331080374,
 '1037': 0.002900162669344578,
 '8482': 0.0028063568687188775,
 '483': 0.0032597515717430973,
 '3567': 0.0001368001259124800 9,
 '6172': 0.002837625468927444,
 '2892': 0.008872465309180852,
 '1362': 0.004901353082692858,
 '787': 0.0024311336662160747,
 '399': 0.005288302010273873,
 '777': 0.00850115068170412,
 '1332': 0.006722749044841879,
 '318': 0.0005276576285195661,
 '769': 0.0003478631773203065,
...
 '1016': 6.644577544320461e-05,
 '5118': 0.000836435055579164,
 '7499': 3.126860020856688e-05,
 '7078': 0.0003361374522420939 5,
 ...}
```

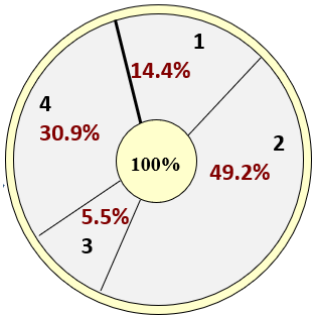
ε) ΣΥΝΑΡΤΗΣΗ ΚΑΤΑΛΛΗΛΟΤΗΤΑΣ

Η επιλογή των ατόμων γίνεται μέσω της συνάρτησης καταλληλότητας (fitness function) μέσω της οποίας θα βρίσκουμε μία τιμή για κάθε άτομο και μέσω αυτόν θα βρίσκουμε τα καταλληλότερα άτομα. Από την εκφώνηση προτείνονται οι λέξεις που γονίδια που σχηματίζουν είναι πιο σημαντικές όπου για αυτό θα χρησιμοποιήσουμε την tf-idf.Ενώ για την δεύτερη προτεινόμενη θα εφαρμόσουμε μία ποινή στα άτομα που έχουν υψηλό αριθμό εμφάνισης στο κείμενο.

Η συνάρτηση καταλληλότητας παίρνει σαν όρισμα το άτομο .Στην πρώτη for ,για κάθε θέση στο άτομο εάν η τιμή που επιλέχθηκε είναι μικρότερη από 1(από το new_list που περιέχει το tf_idf), τότε πρόσθεσε στο fitness την τιμή αυτή. Επίσης θέτουμε μια ποινή κατώτατο όριο λέξεων 1000,όπως μας ζητείται από την εκφώνηση.

```
def getFitness(individual):
    counter = 0
    fitness = 0

    #gia kathe thesi sto indi
    for word_index in individual:
        #vale sto string thn antistoixi leksi
        if new_list[word_index] < 1:
            fitness += new_list[word_index]
    #penalty for >1000 times of 1
    counter = 0
    for word_index in individual:
        if word_index==1:
            counter+=1
    if counter<1000:
        fitness -= 10
    else:
        pass

    return fitness,
```

στ) ΓΕΝΕΤΙΚΟΙ ΤΕΛΕΣΤΕΣ



I) Ρουλέτα με βάση το κόστος: Αποτελεί την πιο συχνή μέθοδο για την επιλογή ατόμων. Πιο συγκεκριμένα χρησιμοποιείται μία ρουλέτα η οποία περιέχει σχισμές και το μέγεθος κάθε σχισμής είναι ανάλογο της απόδοσης που έχει κάθε άτομο να επιλεγεί. Όταν ένα άτομο είναι πιο κατάλληλο δηλαδή έχουν μεγαλύτερο fitness

value τότε καταλαμβάνουν μεγαλύτερη σχισμή στην ρουλέτα, άρα και μεγαλύτερη πιθανότητα να επιλέγει.

Ρουλέτα με βάση την κατάταξη: Γίνεται χρήση της όταν τα άτομα έχουν όμοια πιθανότητα να επιλεγούν οπότε και παρόμοια σχισμή στην ρουλέτα .Το αρνητικό με αυτή την μέθοδο είναι ότι δεν εστιάζει στα άτομα που έχουν μεγάλο fitness value ,άρα θα υπάρχουν κακές επιλογές γονέων.

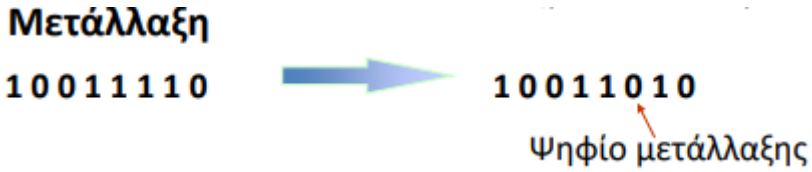
Τουρνουά: Στην επιλογή μέσω τουρνουά επιλέγονται τυχαία ένας αριθμός ατόμων και έπειτα από τα άτομα που επιλέχθηκαν, κρατιούνται μόνο τα καλύτερα για να γίνουν γονείς. Με την μέθοδο αυτή δεν επιλέγονται ποτέ άτομα που είναι χειρότερα από όσα έχουν επιλεχθεί άρα θα έχουμε πιο γρήγορη σύγκλιση στον αλγόριθμο μας αφού θα έχουμε καλές λύσεις σε κάθε περίπτωση. Για το πρόβλημα που καλούμαστε να επιλύσουμε επέλεξα αυτή την μέθοδο, καθώς στην πρώτη μέθοδο χρησιμοποιούμε συνέχεια ίδια άτομα καθώς έχουν μεγαλύτερη πιθανότητα επιλογής ενώ στην δεύτερη δεν υπάρχει εστίαση στο fitness value του ατόμου.

II) Διασταύρωση μονού σημείου: Με την χρήση αυτής, γίνεται τυχαία επιλογή σε ένα σημείο διασταύρωσης και στους δύο γονείς. Τα δυαδικά ψηφία που υπάρχουν δεξιότερα του σημείου αυτού που επιλέχθηκε ανταλλάσονται μεταξύ των γονέων .Έτσι από το crossover αυτό προκύπτουν 2 παιδία που έχουν γενετική πληροφορία και από τους 2 γονείς.

Διασταύρωση πολλαπλού σημείου: Με την χρήση αυτής επιλέγονται τυχαία σημεία για διασταύρωση από τους δύο γονείς. Τα δυαδικά ψηφία που δεν επιλέχθηκαν και βρίσκονται ενδιάμεσα και εξωτερικά ανταλλάσονται μεταξύ των γονέων. Με την χρήση αυτής της μεθόδου διορθώνεται το πρόβλημα που υπάρχει στην διασταύρωση μονού σημείου όπου χάνεται πολύτιμη πληροφορία από τους γονείς με τον τρόπο που γίνεται η επιλογή.

Ομοιόμορφη διασταύρωση: Όπως είναι αναμενόμενο με την μέθοδο αυτή κάθε ψηφίο στην δυαδική ακολουθία έχει την ίδια πιθανότητα να επιλέγει σε σύγκριση με κάθε άλλο ψηφίο. Πιο συγκρεκριμένα αποτελεί μία ενδιάμεση λύση των άλλων 2 μεθόδων που αναλύσαμε παραπάνω .Επίσης συγκλίνει πιο αργά σε σύγκριση με τις άλλες 2 καθώς ασχολείται με όλα τα ψηφία της ακολουθίας των γονέων.

III) Μετάλλαξη:



Όπως παρατηρούμε και στην παραπάνω εικόνα η μετάλλαξη δημιουργεί μια τυχαία αλλαγή στα ψηφία της ακολουθίας. Αυτό γίνεται με την αντιστροφή κάποιον δυαδικών ψηφιών με τυχαίο τρόπο. Όμως επειδή δεν ξέρουμε αν η μετάλλαξη θα μας δώσει μία καλή ακολουθία δυαδικών ψηφίων θα εξετάσουμε την μέθοδο του ελιτισμού ,μέσω του οποίου προκύπτουν άτομα που θα μας δώσουν καλές λύσεις

.Έτσι μία στρατηγική σε εξελικτικούς αλγόριθμους όπου η καλύτερη, μία ή περισσότερες λύσεις, σε κάθε γενιά, εισάγονται στην επόμενη, χωρίς να υποστούν καμία αλλαγή. Αυτή η στρατηγική συνήθως επιταχύνει τη σύγκλιση του αλγορίθμου. Όμως στην άσκηση μας χρησιμοποιούμε δυαδική κωδικοποίηση οπότε δεν θα χρησιμοποιήσουμε την μέθοδο του ελιτισμού.

Β2. ΥΛΟΠΟΙΗΣΗ ΓΑ

Ο αλγόριθμος που υλοποίησα είναι σε γλώσσα python μέσω του jupyter notebook.Η κύρια βιβλιοθήκη που χρησιμοποίησα για την υλοποίηση του γενετικού αλγορίθμου είναι η DEAP.

Η main παίρνει σαν όρισμα το μέγεθος του πληθυσμού , την πιθανότητα διασταύρωσης και μετάλλαξης. Αρχικά ορίζω μια for που θα τρέχει 10 φορές συνολικά ,δηλαδή 10 επαναλήψεις του αλγορίθμου όπως μας ζητείται από την εκφώνηση αφού έχουμε στοχαστικό αλγόριθμο θέλουμε 10 επαναλήψεις για να είμαστε ακριβείς. Στην συνέχεια γίνεται η δημιουργία του πληθυσμού και γίνεται η εκτίμηση της fitness function. Ορίζω το generation=0 , το previous_fit αποθηκεύει την μέγιστη τιμή fitness που έχω πριν εκτελεστεί η while.Η while θα εκτελείται όσο τα criteria=False. Για κάθε iterations τρέχει ο συνολικός αριθμός των generations που θα εκτελεστούν σε κάθε επανάληψη .Για αυτό χρησιμοποιώ τον μετρητή generation.

```
def main(population_size,prob_cross,prob_mutation):


  for i in range(10): # iterations
      print("%d ITERATION "%(i+1))

       # dimiourgia plithismou
      pop = toolbox.population(n=population_size)

    #Evaluation fitness function
      fitnesses = list(map(toolbox.evaluate, pop))
      for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit



    # CXPB pithanotita zeugaromatos
    # MUTPB pithanotita metalakseis
      CXPB, MUTPB = prob_cross, prob_mutation

      #Statistics
      stats = tools.Statistics(key=lambda ind:ind.fitness.values)
      record = stats.compile(pop)
      log = tools.Logbook()
      log.record(gen=0, **record)

        # Extracting all the fitnesses of (epistrefi to fitness)
      fits = [ind.fitness.values[0] for ind in pop]

    # Variable keeping track of the number of generations
    #metritis gia generation
      generation=0
      #Save to fitness pou exoume
      previous_fit=max(fits)
      #best fitness apo oles tis genies
      bestfitness=0
  #creteria
      g = 1
      critiria = False
      max_g=50
      fitness_unchanged = 0

  # Begin the evolution
      while critiria==False :
        generation +=1

    # A new generation
        print("-- Generation %i --" % generation)

    # Select the next generation individuals
        offspring = toolbox.select(pop, len(pop))
    # Clone the selected individuals
        offspring = list(map(toolbox.clone, offspring))

    # Apply crossover and mutation on the offspring
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
```

Έχουμε δημιουργήσει τα εξής κριτήρια ώστε να συνεχίσει να τρέχει η while.Εάν το g>=max_g τότε κάνε τα critira=True ώστε να σταματήσει η εκτέλεση του αλγορίθμου για το συγκεκριμένο iteration.Αλλιώς εάν το g>25 γενιές και το max(fit) που έχουμε τώρα είναι μικρότερο από το (1.001*previous_fit) που είχαμε πριν(δηλαδή βελτιώνεται κάτω από ένα ποσοστό) τότε πάλι critiria true και σταματάει η εκτέλεση για το iteration.

Αλλιώς εάν το g>25 και η τιμή του προηγουμένου fitness είναι ίσο με το τωρινό fitness δηλαδή δεν υπάρχει κάποια βελτίωση τότε αύξησε τον μετρητή fitness_unchanged κατά 1 κάθε φορά εάν συμβεί αυτό 5 φορές δηλαδή δεν υπάρξει

κάποια βελτίωση στο fitness κάνε το critiria=True και σταμάτα την εκτέλεση του αλγορίθμου στο συγκεκριμένο iteration.

Αλλιώς σε κάθε άλλη περίπτωση συνεχίζει κανονικά η εκτέλεση της while,αυξάνει η γενιά και το fitness_unchanged=0.

```
# Apply crossover and mutation on the offspring
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
        if random.random() < CXPB:
            toolbox.mate(child1, child2)
            del child1.fitness.values
            del child2.fitness.values

    for mutant in offspring:
        if random.random() < MUTPB:
            toolbox.mutate(mutant)
            del mutant.fitness.values


# Evaluate the individuals with an invalid fitness
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

# Replace the old population by the offspring
    pop[:] = offspring


# Gather all the fitnesses in one list and print the stats
    fitness = [ind.fitness.values[0] for ind in pop]

    record = stats.compile(pop)
    log.record(gen=g, **record)
    best = log.select("max")

#CRETERIA FOR WHILE LOOP(OSO TO TERM=FALSE trexei h while)
    # reached max number of generations
    if g >= max_g:
        critiria = True
        print("<-------------------ITERATION %d COMPLETED ----------------->" %(i+1))
    # best individual of gen is <1% better than best individual of previous gen
    elif (g > 25) and max(fits) < (1.001*previous_fit):
        critiria = True
        print("<-------------------ITERATION %d COMPLETED ----------------->" %(i+1))
    # best individual of gen is same as best individual of previous gen
    elif (g > 25) and previous_fit == max(fits):
        fitness_unchanged += 1
        #kan g>25 kai exw 5 fores stasimo fitness tote critiria=True
        if fitness_unchanged >= 5:
            critiria = True
            print("<-------------------ITERATION %d COMPLETED ----------------->" %(i+1))
        else:
            g += 1
    # else continue
    else:
        fitness_unchanged = 0
        g += 1
```

Β3. ΑΞΙΟΛΟΓΗΣΗ ΚΑΙ ΕΠΙΔΡΑΣΗ ΠΑΡΑΜΕΤΡΩΝ

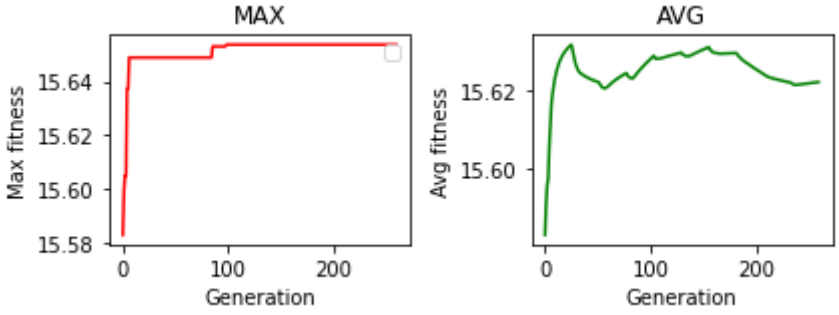
α)

Α/Α	ΜΕΓΕΘΟΣ ΠΛΗΘΥΣΜΟΥ	ΠΙΘΑΝΟΤΗΤΑ ΔΙΑΣΤΑΥΡΩΣΗΣ	ΠΙΘΑΝΟΤΗΤΑ ΜΕΤΑΛΛΑΞΗΣ	ΜΕΣΗ ΤΙΜΗ ΒΕΛΤΙΣΤΟΥ	ΜΕΣΟΣ ΑΡΙΘΜΟΣ ΓΕΝΕΩΝ
1	20	0.6	0.00	15.65	14.46
2	20	0.6	0.01	15.68	14.46
3	20	0.6	0.10	15.70	14.46
4	20	0.9	0.01	15.71	14.46
5	20	0.1	0.01	15.65	14.46
6	200	0.6	0.00	15.83	14.46
7	200	0.6	0.01	15.82	14.46
8	200	0.6	0.10	15.85	14.46
9	**200**	**0.9**	**0.01**	**15.87**	**14.46**
10	200	0.1	0.01	15.72	14.46

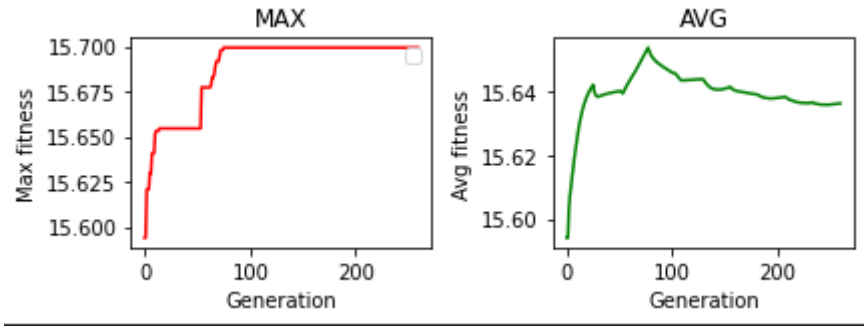
Τα πειράματα εκτελέστηκαν αρκετές φορές καθώς δεν είναι σίγουρο ότι θα καταλήξουν στην ίδια απόδοση. Παρατηρούμε το καλύτερο αποτέλεσμα εμφανίζεται στην περίπτωση 9, με μέγεθος πληθυσμού 200,πιθανότητα διασταύρωσης 0.9 και πιθανότητα μετάλλαξης 0.01.Οπότε στο τελευταίο ερώτημα θα γίνει η χρήση του γενετικού αλγορίθμου με βάση τις τιμές που μας δίνουν το βέλτιστο αποτέλεσμα στον ίδιο αριθμό γενεών.

β)

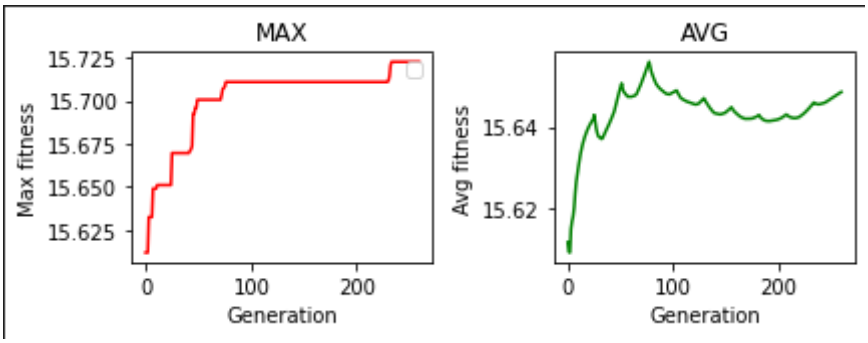
Population:20 , Crossover:0.6,Mutation:0.00



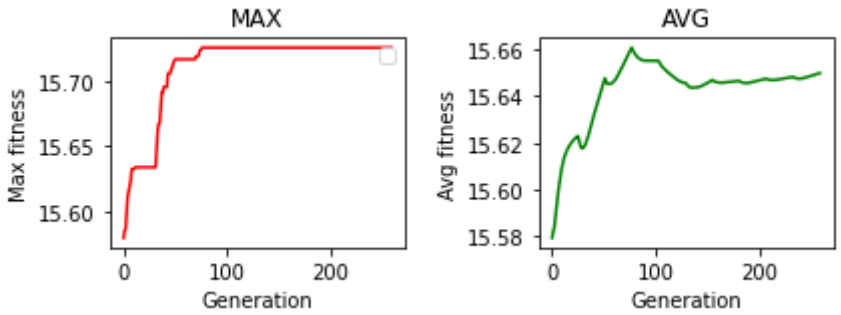
Population:20 , Crossover:0.6,Mutation:0.01



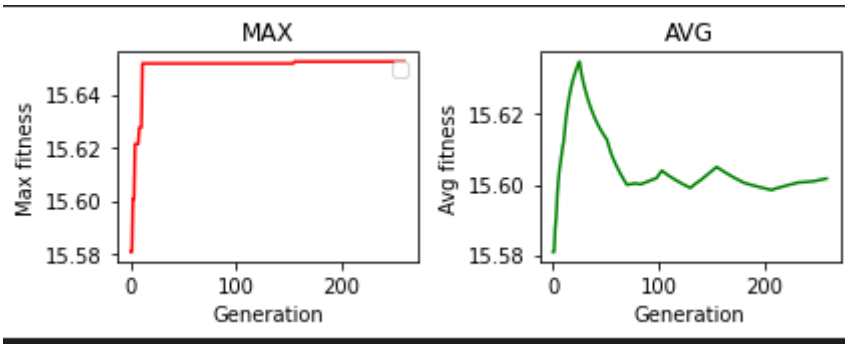
Population:20 , Crossover:0.6,Mutation:0.10



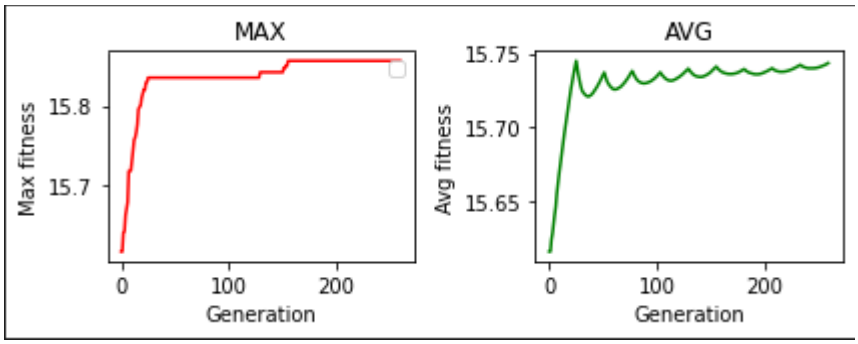
Population:20 , Crossover:0.9,Mutation:0.01



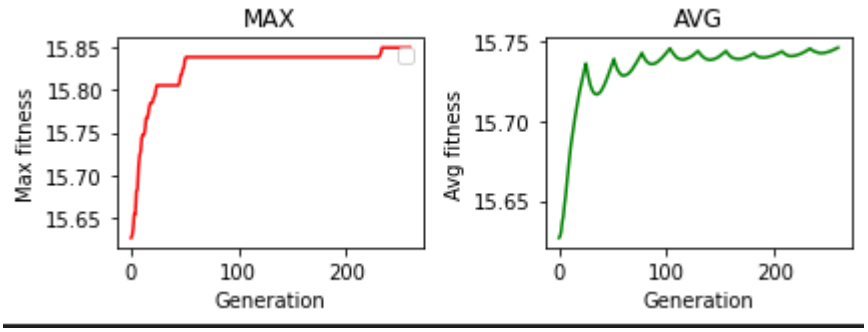
Population:20 , Crossover:0.1,Mutation:0.01



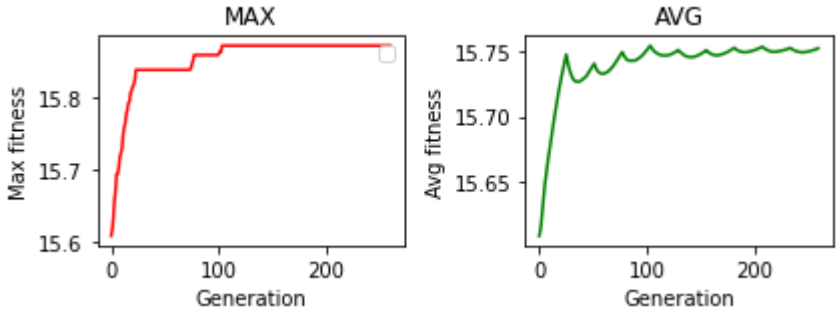
Population:200 , Crossover:0.6,Mutation:0.00



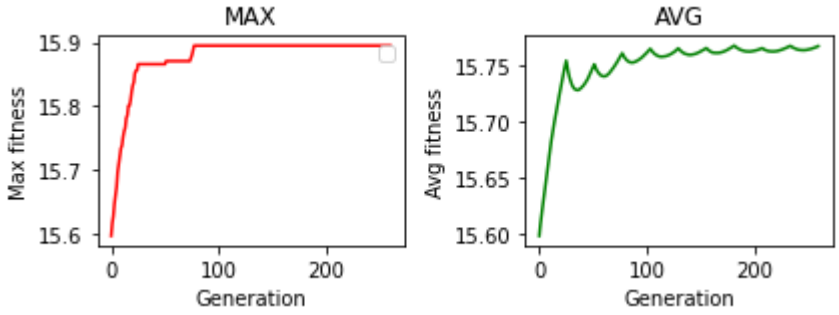
Population:200 , Crossover:0.6,Mutation:0.01



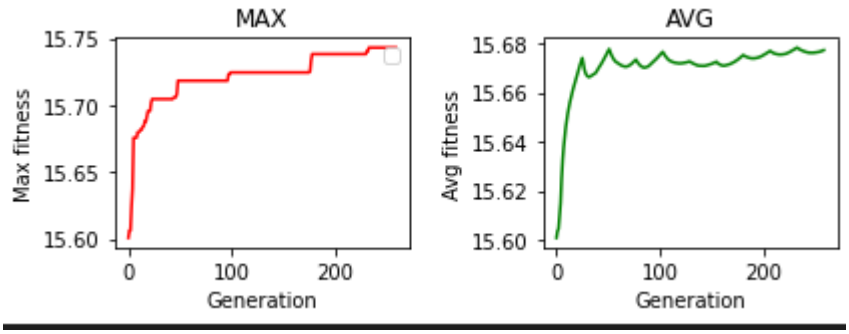
Population:200 , Crossover:0.6,Mutation:0.10



Population:200 , Crossover:0.9,Mutation:0.01



Population:200 , Crossover:0.1,Mutation:0.01



γ) Παρατηρούμε ότι όσο αυξάνουμε το μέγεθος του πληθυσμού ο αλγόριθμος αργεί περισσότερο να εκτελεστεί , που σημαίνει πιο αργή σύγκλιση σε σχέση με την περίπτωση που ο πληθυσμός είναι 20.Η πιο αργή σύγκλιση οδηγεί σε καλύτερα αποτελέσματα, άρα προτιμάμε πληθυσμό 200 και ας χρειάζεται παραπάνω χρόνο να ολοκληρωθεί η εκτέλεση του γενετικού αλγορίθμου. Επίσης παρατηρούμε στις περισσότερες καταστάσεις όσο αυξάνεται η πιθανότητα μετάλλαξης τόσο καλύτερες τιμές παίρνουμε στην fitness function. ‘Αρα καταλαβαίνουμε ότι η πιθανότητα μετάλλαξης βοήθησε το γενετικό στην δημιουργία καλύτερων λύσεων σε σύγκριση με πριν .Το ίδιο ισχύει και για την πιθανότητα διασταύρωσης παρατηρούμε ότι στην περίπτωση 9: με μεγαλύτερη πιθανότητα διασταύρωση και μετάλλαξη σε σύγκριση με την περίπτωση 10 παίρνουμε καλύτερα αποτελέσματα ,ενώ στις υπόλοιπες περιπτώσεις που έχουμε την μικρότερη πιθανότητα διασταύρωσης η τιμή της συνάρτησης καταλληλότητας μειώνεται .Αυτό είναι λογικό αφού δεν δημιουργούνται νέα άτομα καλύτερα σε σύγκριση με πριν.

Για αυτό θα επιλέξω τις συγκεκριμένες τιμές ώστε να εξετάσουμε στο επόμενο ερώτημα το νευρωνικό δίκτυο.

9	**200**	**0.9**	**0.01**	**15.87**	**14.46**

Β4. ΕΠΙΛΟΓΗ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΤΝΔ

Retrained Model

α)

Όπως βλέπουμε αφού αποθήκευσα στον πίνακα X_train τα δεδομένα που πήρα από τον γενετικό αλγόριθμο,τα πέρασα την συνάρτηση train_test_split ώστε να τα διαχωρίσω σε δεδομένα train και testing.Παρατηρούμε ότι έχουμε (5755 δεδομένα train και 2476 test).

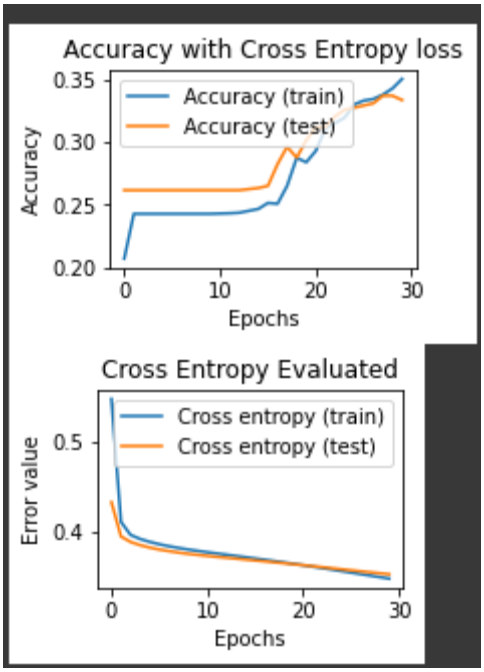
```
X_train.shape
for i in range(len(keep)):
    if keep[i] == 0:
        X_train[:,i] = 0


X_train, X_test, Y_train, Y_test = train_test_split(X_train, y, test_size=0.3, random_state=0)

print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

```
(5775, 8522) (2476, 8522) (5775, 20) (2476, 20)
```

Έπειτα φόρτωσα το μοντέλο ce.h5 αφού διαμόρφωσα κατάλληλα το μοντέλο ώστε να δέχεται εισόδους με το shape των δεδομένων που έχουμε έκανα τις μετρήσεις που χρειαζόταν.Έχω χρησιμοποιήσει μόνο το cross-entropy , και όχι το mse model καθώς στην πρώτη εργασία είδαμε ότι το ce λειτουργεί καλύτερα για το πρόβλημα μας.

```
import tensorflow as tf
from tensorflow import keras
from keras.models import load_model
model = tf.keras.models.load_model("ce.h5")
```

Στην πρώτη εργασία όσο αφορά το νερωνικό δίκτυο είδαμε οτί η ικανοτήτα γενίκευσης ήταν αρκετά καλή και δεν είχαμε πρόβλημα με το overfitiing.Σε αυτή την περίπτωση παρατηρούμε ότι το μοντέλο που έχουμε με την χρήση των δεδομένων από τον γενετικό αλγόριθμο πετυχαίνει καλύτερο accuracy σε σχέση με το το νευρωνικό δίκτυο που είχαμε στην πρώτη εργασία. Παρατηρούμε όμως στα δεδομένα υπάρχει overfitting μέσω της καμπύλης που βλέπουμε. Επίσης και στις 2 περίπτωσης παρόλο που ο γενετικός δείχνει καλύτερη ακρίβεια σε σχέση με το νευρωνικό που χρησιμοποιήσαμε την τεχνική bow,βλέπουμε ότι η μείωση τον δεδομένων από (8251) input στο νευρωνικό έχουμε στο νέο μοντέλο τα μισά, η ακριβεία είναι χαμηλότερη σε σχέση με την πρώτη εργασία.




```
Epoch 10/50
181/181 [==============================] - 12s 67ms/step - loss: 0.2808 - accuracy: 0.3280
Epoch 11/50
181/181 [==============================] - 12s 68ms/step - loss: 0.2778 - accuracy: 0.3311
Epoch 12/50
181/181 [==============================] - 12s 68ms/step - loss: 0.2743 - accuracy: 0.3316
Epoch 13/50
...
181/181 [==============================] - 12s 67ms/step - loss: 0.1976 - accuracy: 0.3730
78/78 [==============================] - 2s 19ms/step - loss: 0.3408 - accuracy: 0.2342
Accuracy for  retrained model is 0.2342487871646881
Lose for retrained model is  0.3408249020576477
```

Non Retrained Model

```
loss, accuracy = model.evaluate(X_test_new, Y_test, verbose=0)
print("Accuracy for non retrained model is", accuracy)
print("loss for non retrained model is", loss)
```

[23] ✓ 1.5s

```
Accuracy for non retrained model is 0.1676090508699417
loss for non retrained model is 0.82370525598526
```

Β)Όπως προαναφέρθηκε και στο ερώτημα α, βλέπουμε ότι η χρήση δεδομένων από τον γενετικό πετυχαίνει καλύτερη απόδοση όμως υπάρχει εμφάνιση overfitting λόγω της μείωσης των δεδομένων (train_data και test_data).