Τμημα Μηχανικων Ηλεκτρονικων Υπολογιστων και Πληροφορικης

Πολυτεχνικη Σχολη

Πολυδιάστατες Δομές Δεδομένων - Εργασία Εξαμήνου

Μία εργασία

του

Νικόλαου Σκαμνέλου

A.M: 1041878

Έτος: 8ο

Πάτρα 19/2/2022

Περιεχόμενα

1 Εισαγωγή **2**

2 Ανάλυση της συλλογής **3**
2.1 Η Συλλογή Cystic Fibrosis 3
2.2 Ανάλυση των κειμένων και των ερωτημάτων της συλλογής 3

3 Δημιουργία Δέντρων **10**
3.1 Εισαγωγή 10
3.2 Quad Tree 10
3.2.1 Οι κόμβοι του Octree 11
3.2.2 Η κλάση του Octree 12
3.2.3 k-Nearest-Neighbors στο Octree 16
3.3 K-d Tree 17
3.3.1 Οι κόμβοι του K-d Tree 18
3.3.2 Η κλάση του K-d Tree 18
3.3.3 k-Nearest-Neighbors στο K-d Tree 20
3.4 R-tree 23
3.4.1 Οι κόμβοι του R-tree 23
3.4.2 Η κλάση του R-tree 24
3.4.3 k-Nearest-Neighbors στο R-tree 28

4 Επίλογος **29**

Chapter 1

Εισαγωγή

Το παρόν έγγραφο αποτελεί την αναφορά της εργασίας εξαμήνου του Νικόλαου Σκαμνέλου για το μάθημα με τίτλο "Πολυδιάστατες Δομές Δεδομένων". Για τα πειράματα χρησιμοποιήθηκε η συλλογή Cystic Fibrosis (Shaw et al., 1991). Περισσότερες πληροφορίες ακολουθούν στο επόμενο κεφάλαιο. Η διαδικασία που ακολουθήθηκε με λίγα λόγια είναι η εξής. Αρχικά, αναλύθηκαν τα κείμενα και τα ερωτήματα της συλλογής, και δημιουργήθηκαν με την χρήση του αλγορίθμου apriori Agrawal and Srikant, 1994 και του Set-based μοντέλου Possas et al., 2002, διανύσματα ερωτημάτων και κειμένων. Στην συνέχεια χρησιμοποιήθηκε η τεχνική PCA Maćkiewicz and Ratajczak, 1993 για την μείωση των διαστάσεων των παραπάνω διανυσμάτων. Σκοπός της παραπάνω διαδικασίας ήταν η αναπαράσταση των κειμένων ως σημεία στο τρισδιάστατο χώρο. Με τα σημεία που αντιστοιχούν σε κείμενα δημιουργήθηκαν δομές δεδομένων με την μορφή δέντρων. Πιο συγκεκριμένα δημιουργήθηκαν Quad Trees, K-d Trees και R-Trees. Τέλος, με την βοήθεια αλγορίθμων τύπου k-Nearest-Neighbors (kNN) και των σημείων που αντιστοιχούν στα ερωτήματα απαντήθηκαν διάφορα ερωτήματα ομοιότητας. Σημειώνεται ότι, μετά από συζήτηση με τον υπεύθυνο καθηγητή κ. Σιούτα Σπυρίδων, δόθηκε η επιλογή μη χρήσης της μεθόδου LSH, ακόμα και αν ζητείται από την εκφώνηση.

Chapter 2

Ανάλυση της συλλογής

2.1 Η Συλλογή Cystic Fibrosis

Η συλλογή που χρησιμοποιήθηκε είναι η Cystic Fibrosis (Shaw et al., 1991), οι οποία περιέχει 1209 κείμενα και 100 ερωτήματα. Συνολικά έχει μέγεθος 1.47 Megabyte. Τα κείμενα της συλλογής είναι σχετικά μικρά (5 λέξεις) έως μέτρια (500 λέξεις) σε μέγεθος. Κατά την ανάλυση των κειμένων της συλλογής αφαιρέθηκαν οι αγγλικές ασήμαντες λέξεις, όπως αυτές ορίζονται από την βιβλιοθήκη NTLK (Steven Bird and Loper, 2009). Συνολικά αφαιρέθηκαν 127 λέξεις.

2.2 Ανάλυση των κειμένων και των ερωτημάτων της συλλογής

Επόμενο βήμα είναι η ανάλυση των κειμένων και των ερωτημάτων της συλλογής με σκοπό την δημιουργία των απαιτούμενων διανυσμάτων που αναφέρθηκαν στην εισαγωγή. Στο 2.1 φαίνεται η συνολική διαδικασία που ακολουθήθηκε.

```
1  from getQueries import *
2  from getRelevant import *
3  from AnalyzeCollectionDocs import *
4  import nltk
5  from nltk.corpus import stopwords
6  import string
7  from sklearn.decomposition import PCA
8
9  def DocumentReprsentationAsPoints():
10
11     #Analyze the collection
12     postingList, collectionTerms, docsLen, fileList = AnalyzeCollection
           ()
13     #print(postingList)
14     #print(fileList)
15     print("Done Analizing the documents")
16
17     #Get and preprocess the list of Stopwords
18     stop_words = set(stopwords.words('english'))
19     table = str.maketrans('', '', string.punctuation)
```

```
20      stop_words = [w.translate(table) for w in stop_words]
21      stop_words = [word.upper() for word in stop_words]
22
23
24
25      #Preprocess the Query and get the keywords
26      Q = "CF Fibrosis Cystic Patients Effects Properties"
27      print('Keywords ========= ',Q)
28      Q = Q.upper()
29      Q = Q.split()
30      keywords = [w for w in Q if not w in stop_words]
31      keywords = list(set(Q))  # to ignore duplicate words as queries
32      #print(keywords)
33      numOfTermsets = (2 ** (len(keywords))) - 1
34
35      #Generate the Termesets of the Keywords using apriori
36      #print(len(collectionTerms))
37      One_termsets = one_termsets(keywords, collectionTerms, postingList,
            0)
38      #print(One_termsets)
39
40      final_list = apriori(One_termsets,0)
41      #print(final_list)
42
43      #Calculate the termset frequency for every doc
44      docs, doc_vectors = fij_calculation(fileList, final_list,
            postingList, collectionTerms)
45      #print(len(doc_vectors))
46
47      #Calculate the inverse document frequency of every termset
48      idf_vec = calculate_idf(final_list, len(fileList))
49
50      #Calculate the tf*idf for every document
51      documentmatrix = doc_rep(doc_vectors, idf_vec)
52      #print(len(documentmatrix))
53
54      #Transform the document matrix into a numpy array and run pca to
            reduce dimensions to 3 using pca
55      documentMatrix = numpy.array(documentmatrix)
56      #print(documentMatrix)
57      #print(len(documentMatrix[1]))
58
59      pca = PCA(n_components=3)
60      transformedDocumentMatrix = pca.fit_transform(documentMatrix, y=
            None)
61      #print(transformedDocumentMatrix)
62
63      return transformedDocumentMatrix
```

Listing 2.1: Representing the documents and queries as points

Αρχικά, χρησιμοποιήθηκε μία συνάρτηση με όνομα AnalyzeCollection() (γραμμές 11-12), η οποία για κάθε κείμενο και ερώτημα της συλλογής, εξάγει τους όρους του, το που εμφανίζεται και στην συχνότητα εμφάνισης του και τα αποθηκεύει σε μία λίστα, που ονομάζεται postingList. Επίσης, αποθηκεύει τους όρους της συλλογής (collectionTerms), τα μεγέθη των κειμένων της συλλογής (docsLen) και τα ονόματα των κειμένων και ερωτημάτων της συλλογής (fileList). Τα κείμενα έχουν όνομα με την μορφή "txtfiles//1-1209", ενώ τα ερωτήματα έχουν όνομα με την μορφή "txtfiles//1210-1309". Η μορφή της postingList είναι η εξής:

["Όρος 1" , [Αρχείο k, Συχνότητα k, ..., Αρχείο m, Συχνότητα m],..., "Όρος 2", [Αρχείο x, Συχνότητα x, ..., Αρχείο y, Συχνότητα y]]

Η υλοποίησή της παραπάνω συνάρτησης φαίνεται στο 2.2. Σημειώνεται ότι τα αρ-

χεία είναι αποθηκευμένα ως αρχεία σε ένα φάκελο με όνομα "txtfiles", ενώ τα ερωτήματα προέρχονται από μία συνάρτηση με όνομα getQueries(), η οποία επιστρέφει τα ερωτήματα της συλλογής με μορφή πίνακα, του οποίου κάθε θέση περιέχει ένα ερώτημα.

```
1  def AnalyzeCollection():
2      file_list = []
3      for item in os.listdir('txtfiles'):
4          name = os.path.join("txtfiles", item)
5          if os.path.getsize(name) == 0:
6              print('%s is empty:' % name)
7              os.remove(name)
8          else:
9              file_list.append([name, os.path.getsize(name)])
10     file_list = sorted(file_list, key=itemgetter(1), reverse=True)
11
12     postinglist =[]
13     collection_terms = []
14     doc_length =[]
15
16     for file in file_list:
17         filename = file[0]
18         #print(file)
19         with open(filename, 'r') as fd:
20         #list containing every word in text document
21             text = fd.read().split()
22             temp = getPostingList(filename,text,postinglist,
                   collection_terms)
23             postinglist = temp[0]
24             collection_terms = temp[1]
25             doc_length.append(temp[2])
26             #print('Analyzed File:' + filename)
27     queries = getQueries()
28     counter = 1210
29     for query in queries:
30         qname = "txtfiles\\" + str(counter)
31         file_list.append([qname, 15])
32         text = query.split()
33         text = [word.upper() for word in text]
34         temp = getPostingList(qname,text,postinglist,collection_terms)
35         postinglist = temp[0]
36         collection_terms = temp[1]
37         doc_length.append(temp[2])
38         counter+=1
39
40     return postinglist, collection_terms, doc_length, file_list
41
42 def getPostingList(name,text, postingl,collection_terms):
43
44     for term in text:
45         if term not in postingl:
46             collection_terms.append(term)
47             postingl.append(term)
48             postingl.append([name, text.count(term)])
49         else:
50             existingtermindex = postingl.index(term)
51             if name not in postingl[existingtermindex + 1]:
52                 postingl[existingtermindex + 1].extend([name, text.
                       count(term)])
53     return postingl, collection_terms, len(text)
```

Listing 2.2: Analyzing the documents and queries

Επόμενο βήμα είναι η προεπεξεργασία των λέξεων κλειδιών που θα χρησιμοποιηθούν στη δεικτοδότηση. Αυτή φαίνεται στις γραμμές 25-32 του 2.1. Ως λέξεις κλειδιά χρησιμοποιήθηκαν η λέξεις "CF", "Fibrosis", "Cystic", "Patients", "Effects"

και "Properties". Στην συνέχεια πρέπει να παραχθούν τα μονοσύνολα όρων για τον αλγόριθμο apriori (γραμμές 33-38). Για να επιτευχθεί αυτό χρησιμοποιείται μία συνάρτηση με όνομα one_termsets(), η οποία δέχεται ως είσοδο τη λίστα με τις λέξεις κλειδιά, την λίστα με τους όρους της συλλογής, την postingList και μία τιμή που λειτουργεί ως ελάχιστη συχνότητα εμφάνισης, και παράγει τα μονοσύνολα. Στο 2.3 φαίνεται η υλοποίησή της. Η συνάρτηση διατρέχει την postingList και αποθηκεύει σε μία λίστα με όνομα One_termsets τις λέξεις κλειδιά, καθώς και τους πίνακες με τα κείμενα που εμφανίζονται οι λέξεις και την συχνότητα εμφάνισης τους, εφόσον ικανοποιούν την συνθήκη ελάχιστης συχνότητας εμφάνισης σε κείμενα.

```
1  def one_termsets(keywords, termns, postinglist, minfreq):
2      One_termsets = []
3      for word in keywords:
4          if word in termns:
5              i = postinglist.index(word)
6              doc = postinglist[(i + 1)]
7              doc = doc[::2]
8              word = [''.join(word)]
9              #print(doc)
10             if len(doc) > minfreq:
11                 One_termsets.append([word, doc])
12         else:
13             print('word %s has not required support or it already
                   exists:' % word)
14     return One_termsets
```

Listing 2.3: The function that generates the one termsets

Έχοντας παράξει τα μονοσύνολα, εκτελείται ο αλγόριθμος apriori (Agrawal and Srikant, 1994) (γραμμή 40). Η υλοποίηση του φαίνεται στο 2.4. Η λειτουργία του είναι αυτή που περιγράφεται στην αντίστοιχη αναφορά. Τα συχνά σύνολα όρων παράγονται ως συνδυασμός των μονοσυνόλων και εφόσον αυτά ικανοποιούν την συνθήκη ελάχιστης συχνότητας αποθηκεύονται σε μία λίστα, η οποία έχει μορφή ίδια με την postingList, με την μόνη διαφορά ότι πλέον αντί όρους της συλλογής, έχει τα παραγόμενα σύνολα όρων.

```
1  def apriori(l1, minfreq):
2      final_list = []
3      final_list.append(l1)
4      k = 2
5      l = l1
6      print('==========Generating frequent sets ==========')
7      while (l != []):
8          c = apriori_gen(l)
9          l = apriori_prune(c, minfreq)
10         final_list.append(l)
11         k += 1
12     return final_list
13
14
15 def apriori_gen(itemset):
16     candidate = []
17     length = len(itemset)
18     for i in range(length):
```

```
19          ele = itemset[i][0]
20          for j in range(i + 1, length):
21              ele1 = itemset[j][0]
22              if ele[0:len(ele) - 1] == ele1[0:len(ele1) - 1]:
23                  c=[]
24                  for k in ele + ele1:
25                      if k not in c:
26                          c.append(k)
27                  candidate.append([c, list(set(itemset[i][1]) & set(
                        itemset[j][1]))])
28      return candidate
29
30  def apriori_prune(termsets_list, min_support):
31      prunedlist = []
32      for j in termsets_list:
33          if len(j[1]) > min_support:
34              prunedlist.append([j[0], j[1]])
35
36      return prunedlist
```

Listing 2.4: The apriori algorithm

Στην συνέχεια πρέπει να υπολογισθούν, χρησιμοποιώντας τα συχνά σύνολα όρων, τα βάρη για το set-based μοντέλο (Possas et al., 2002), τα οποία βασίζονται στην μετρική $TF \times IDF$ (Baeza-Yates and Ribeiro-Neto, 2008). Για το termset frequency (TF) χρησιμοποιείται μία μέθοδο με όνομα fij_calculation() (γραμμές 43-44), που αναλαμβάνει να υπολογίσει τα διανύσματα TF για κάθε σύνολο όρων που έχει παραχθεί από τον αλγόριθμο apriori. Ιδιαίτερη σημασία πρέπει να δοθεί στο γεγονός ότι σύμφωνα με το set-based μοντέλο, σε κάθε θέση ενός διανύσματος κειμένου, αποθηκεύεται η ελάχιστη συχνότητα εμφάνισης μεταξύ των όρων ενός συνόλου όρων στο κείμενο αυτό. Αυτό θα γίνει πιο ξεκάθαρο με ένα παράδειγμα. Έστω το σύνολο όρων Όρος 1, Όρος 2, Όρος 3, και κάθε όρος του συνόλου εμφανίζεται σε ένα κείμενο D, με συχνότητα εμφάνισης 2, 4 και 7 αντίστοιχα. Στο διάνυσμα που αντιστοιχεί στο κείμενο D, στην θέση που αντιστοιχεί στο συγκεκριμένο σύνολο όρων, θα αποθηκευτεί η ελάχιστη τιμή εμφάνισης των όρων του συνόλου, δηλαδή το 2. Η υλοποίηση αυτής της διαδικασία φαίνεται στο 2.5.

```
1   def fij_calculation(file_list, final_list, plist, trms):
2       docs =[]
3       doc_list = []
4       weight_doc_matrix = []
5       doc_vec = []
6       for itemsetList in final_list:
7           for itemset in itemsetList:
8               for file in file_list:
9                   #print(file[0])
10                  docs.append(file[0])
11                  if file[0] in itemset[1]:
12                      itemsetTerms = itemset[0]
13                      for itemset_term in itemsetTerms:
14                          for j in range(1,len(plist),2):
15                              if itemset_term in plist[j-1]:
16                                  if file[0] in plist[j]:
17                                      file_index = plist[j].index(file
                                            [0])
18                                      weight =  plist[j][file_index+1]
19                                      weight_doc_matrix.append(weight)
```

```
20                      doc_vec.append(min(weight_doc_matrix))
21                      weight_doc_matrix = []
22                  else:
23                      doc_vec.append(0)
24              doc_list.append(doc_vec)
25              doc_vec = []
26      doc_list = numpy.transpose(doc_list)
27      doc_list = doc_list.tolist()
28      return docs, doc_list
```

Listing 2.5: Termset frequency calculation

Για το IDF η διαδικασία είναι αρκετά πιο απλή. Για κάθε σύνολο όρων υπολογίζεται από την λίστα εμφάνισης του, ο αριθμός των κειμένων στα οποία εμφανίζεται. Αυτό επιτυγχάνεται με μία συνάρτηση που ονομάζεται calculate_idf() (γραμμές 47-48). Η υλοποίησή αυτής της συνάρτησης φαίνεται στο 2.6.

```
1  def calculate_idf(termsetsL, numofdocs):
2      idf_vector = []
3      for ts in termsetsL:
4          for item in ts:
5              Nt = len(item[1])
6              N = numofdocs
7              if Nt != 0:
8                  idf = log(1 + (N / Nt))
9                  idf_vector.append(idf)
10             else:
11                 idf_vector.append(0)
12     return idf_vector
```

Listing 2.6: Termset document frequency calculation

Το μόνο που μένει να γίνει είναι να συνδυαστούν οι πίνακες με τις συχνότητες των συνόλων όρων και τις συχνότητες κειμένων των συνόλων όρων, και να υπολογισθεί στο $TF \times IDF$. Υπολογίζεται σύμφωνα με τον τύπο που περιγράφεται από το set-based μοντέλο, δηλαδή από τον παρακάτω τύπο:

$$w_{i,j} = sf_{i,j} \times ids_i = sf_{i,j} \times \log \frac{N}{ds_i} \tag{2.1}$$

Όπου το $sf_{i,j}$ είναι ο αριθμός των εμφανίσεων του συνόλου i στο κείμενο j, ids_i είναι η ανάστροφη συχνότητα εμφάνισης του συνόλου i στη συλλογή και το N είναι ο αριθμός των κειμένων στην συλλογή. Η υλοποίηση του παραπάνω φαίνεται στο 2.7.

```
1  def doc_rep(doc_vec, idf_vec):
2      test = numpy.zeros((len(doc_vec), len(idf_vec)))
3      for i in range(len(doc_vec)):
4          for j in range(len(idf_vec)):
5              if doc_vec[i][j] > 0:
6                  test[i][j] = (1 + log(doc_vec[i][j])) * idf_vec[j]
7              else:
8                  test[i][j] = 0
9      return test
```

Listing 2.7: $TF \times IDF$ calculation

Από την παραπάνω διαδικασία παράγονται 1309 διανύσματα, με αριθμό θέσεων

ίσο με τα σύνολα όρων, το καθένα. Αυτό σημαίνει ότι για τις λέξεις κλειδιά που χρησιμοποιήθηκαν, έχουμε διανύσματα 64 θέσεων. Για να μειωθεί η διστακτικότητά των διανυσμάτων χρησιμοποιείται η τεχνική PCA (Maćkiewicz and Ratajczak, 1993) (γραμμές 59-61) και οι διαστάσεις μειώνονται στις τρεις. Αυτή η μείωση ωστόσο δεν είναι χωρίς τίμημα, αφού ένα μέρος της πληροφορίας των διανυσμάτων χάνεται ανεπιστρεπτί.

Chapter 3

Δημιουργία Δέντρων

3.1 Εισαγωγή

Έχοντας παράξει τα τρισδιάστατα διανύσματα κειμένων, είμαστε σε θέση να δημιουργήσουμε τα επιθυμητά δέντρα. Όπως αναφέρθηκε και στην εισαγωγή της εργασίας, υλοποιήθηκαν τρία είδη δέντρων, το Quad Tree, το K-d Tree και το R-tree. Πριν προχωρήσουμε στις κλάσεις που περιγράφουν τα δέντρα ωστόσο, πρέπει να γίνει αναφορά στην κλάση που περιγράφει τα σημεία στο χώρο. Η κλάση αυτή είναι εξαιρετικά απλή και φαίνεται στο 3.1.

```
1 #Class for the points
2 #The dimensions hold the values of the k dimensions of a point
3 class Point:
4     def __init__(self, dimensions):
5         self.dimensions = dimensions
```

Listing 3.1: Point Class

Για ένα σημείο στον χώρο δημιουργείται ένα αντικείμενο τύπου Point, το οποίο έχει διαστάσεις τις διαστάσεις του σημείου.

Στην συνέχεια, θα γίνει ανάλυση των μεθόδων κατασκευής κάθε δέντρο, όπως και διάφορων άλλων λοιπών συναρτήσεων.

3.2 Quad Tree

Σε αυτό το υπό-κεφάλαιο περιγράφονται οι μέθοδοι κατασκευής του Quad Tree, καθώς και των μεθόδων εντοπισμού των k κοντινότερων γειτόνων, δοθέντος ενός σημείου στο χώρο. Το Quad Tree που υλοποιήθηκε είναι τρισδιάστατο οπότε είναι πρακτικά μία από της παραλλαγές του, που ονομάζεται octree. Οι μεθόδοι που περιγράφονται σε αυτό το υπό-κεφάλαιο μπορούν να βρεθούν στο jupyter notebook με όνομα "Quadtree".

3.2.1 Οι κόμβοι του Octree

Αρχικά θα γίνει αναφορά στην κλάση των κόμβων του Octree. Η υλοποίησή της κλάσης αυτής φαίνεται στο 3.2. Κάθε κόμβος του Octree περιγράφει ένα οκτάντα (octant) ενός κύβου. Στην κλάση υπάρχουν οι μεταβλητές για το αρχικό σημείο του κύβου (xdim, ydim, zdim), θεωρώντας ότι αντιστοιχεί στην πάνω, αριστερή και μπροστινή του γωνία. Επίσης, υπάρχουν μεταβλητές με το μέγεθος, σε μονάδες, κάθε διάστασης του κύβου (length, width, height). Τέλος, κάθε κόμβος περιέχει δύο πίνακες, έναν για τα σημεία που περιέχονται στο κύβο - κόμβο - οκτάντα, και ένα που περιέχει αναφορές στα οκτάντα παιδιά του. Αυτά είναι συνολικά 8 για κάθε κόμβο.

```
1  #The quad tree node
2  #x_dim is the starting point of the square in x axis
3  #y_dim is the starting point of the square in y axis
4  #z_dim is the starting point of the square in z axis
5  #length is the length of the square along the z axis
6  #width is the length of the square along the x axis
7  #height is the length of the square along the y axis
8  #points are the points it contains
9  #children are the children nodes it contains (8 in total)
10 class QuadTreeNode():
11     def __init__(self, x_dim, y_dim, z_dim, width, height, length,
       points):
12         self.xdim = x_dim
13         self.ydim = y_dim
14         self.zdim = z_dim
15         self.length = length
16         self.width = width
17         self.height = height
18         self.points = points
19         self.children = []
20
21     #Get the length of the tree
22     def getLength(self):
23         return self.length
24
25     #Get the windth of the tree
26     def getWidth(self):
27         return self.width
28
29     #Get the height of the tree
30     def getHeight(self):
31         return self.height
32
33     #Get the points contained in the node
34     def getPoints(self):
35         return self.points
```

Listing 3.2: The Octree Node Class

3.2.2 Η κλάση του Octree

Προχωρώντας, αναλύεται η κλάση που περιγράφει το ίδιο το octree. Η υλοποίηση της κλάσης αυτής φαίνεται στο 3.3, και είναι αρκετά απλή. Κάθε δέντρο έχεις τρεις σημαντικές μεταβλητές. Πρώτα, μία ακέραια μεταβλητή που δείχνει των μέγιστο αριθμό των σημείων που επιτρέπεται να περιέχει κάθε κόμβος (maximumPoints), τα σημεία (τύπου Point) που περιέχει συνολικά (startingPoints) και τον κόμβο ρίζα (root), ο οποίος είναι ένα αντικείμενο τύπου κλάσης QuadTreeNode και που δέχεται ως μεταβλητές τις συντεταγμένες του αρχικού σημείου του συνολικού κύβου, τις διαστάσεις του, και τα αρχικά σημεία του.

```
1  #Class for the quad tree
2  #maximumPoints are the number of points allowed in a node
3  #startingPoints are the default points at the start
4  #startingX/Y/Z are the x, y, z, starting dimensions
5  #maxX/Y/Z are the maximum x, y, z dimensions on the dataset
6  class QuadTree():
7      def __init__(self, maximumPoints, startingPoints, startingX,
           startingY, startingZ, maxX, maxY, maxZ):
8          self.maximum_points = maximumPoints
9          self.points = startingPoints
10         self.root = QuadTreeNode(startingX, startingY, startingZ,  maxX
               , maxY, maxZ, self.points)
11
12     #Get the points of the tree
13     def getPoints(self):
14         return self.points
15
16     #Divide the tree in 8 different nodes
17     def divideTree(self):
18         treeDivision(self.root, self.maximum_points)
```

Listing 3.3: The Octree Class

Η κλάση QuadTree επίσης περιέχει μία συνάρτηση με όνομα divideTree(), η οποία καλεί μία συνάρτηση που διαχωρίζει το δέντρο σε οκτάντα. Η υλοποίηση αυτής της συνάρτησης φαίνεται στο 3.4.

```
1  #Recursively diviedes the nodes in 8 different pieces -->  Back north
       west (bnw), front north west (fnw), back north east (bne), front
       north east (fne), Back south west (bsw), front south west (fsw),
       back south east (bse), front south east (fse)
2  def treeDivision(nodeToBeDivided, maximumPointsAllowed):
3
4      #If node contains less than allowed points dont divide it
5      if len(nodeToBeDivided.getPoints())<=maximumPointsAllowed:
6          #print(len(nodeToBeDivided.getPoints()))
7          return
8
9      #The new length, width and height
10     l = float(nodeToBeDivided.getLength()/2)
11
12     w = float(nodeToBeDivided.getWidth()/2)
13
14     h = float(nodeToBeDivided.getHeight()/2)
15
16     #Find the points that belong to fsw
17     ptsfsw = []
18     for point in nodeToBeDivided.getPoints():
```

```
19          if point.dimensions[0] >= nodeToBeDivided.xdim and point.
                dimensions[0] <= (nodeToBeDivided.xdim + w) and point.
                dimensions[1]>=nodeToBeDivided.ydim and point.dimensions
                [1]<=(nodeToBeDivided.ydim+h) and point.dimensions[2]>=
                nodeToBeDivided.zdim and point.dimensions[2]<=(
                nodeToBeDivided.zdim+l):
20              ptsfsw.append(point)
21
22      #Create a new node
23      fsw = QuadTreeNode(nodeToBeDivided.xdim, nodeToBeDivided.ydim,
            nodeToBeDivided.zdim, l, w, h, ptsfsw)
24
25      #Divide the node again
26      treeDivision(fsw,maximumPointsAllowed)
27
28      #Find the points that belong to bsw
29      ptsbsw = []
30      for point in nodeToBeDivided.getPoints():
31          if point.dimensions[0] >= nodeToBeDivided.xdim and point.
                dimensions[0] <= (nodeToBeDivided.xdim + w) and point.
                dimensions[1]>=nodeToBeDivided.ydim and point.dimensions
                [1]<=(nodeToBeDivided.ydim+h) and point.dimensions[2]>=
                nodeToBeDivided.zdim+l and point.dimensions[2]<=(
                nodeToBeDivided.zdim+2*l):
32              ptsbsw.append(point)
33
34      #Create a new node
35      bsw = QuadTreeNode(nodeToBeDivided.xdim, nodeToBeDivided.ydim,
            nodeToBeDivided.zdim+l, l, w, h, ptsbsw)
36
37      #Divide the node again
38      treeDivision(bsw,maximumPointsAllowed)
39
40      #Find the points that belong to fnw
41      ptsfnw = []
42      for point in nodeToBeDivided.getPoints():
43          if point.dimensions[0] >= nodeToBeDivided.xdim and point.
                dimensions[0] <= (nodeToBeDivided.xdim + w) and point.
                dimensions[1]>=nodeToBeDivided.ydim+h and point.dimensions
                [1]<=(nodeToBeDivided.ydim+2*h) and point.dimensions[2]>=
                nodeToBeDivided.zdim and point.dimensions[2]<=(
                nodeToBeDivided.zdim+l):
44              ptsfnw.append(point)
45
46      #Create a new node
47      fnw = QuadTreeNode(nodeToBeDivided.xdim, nodeToBeDivided.ydim+h,
            nodeToBeDivided.zdim, l, w, h, ptsfnw)
48
49      #Divide the node again
50      treeDivision(fnw,maximumPointsAllowed)
51
52      #Find the points that belong to bnw
53      ptsbnw = []
54      for point in nodeToBeDivided.getPoints():
55          if point.dimensions[0] >= nodeToBeDivided.xdim and point.
                dimensions[0] <= (nodeToBeDivided.xdim + w) and point.
                dimensions[1]>=nodeToBeDivided.ydim+h and point.dimensions
                [1]<=(nodeToBeDivided.ydim+2*h) and point.dimensions[2]>=
                nodeToBeDivided.zdim+l and point.dimensions[2]<=(
                nodeToBeDivided.zdim+2*l):
56              ptsbnw.append(point)
57
58      #Create a new node
59      bnw = QuadTreeNode(nodeToBeDivided.xdim, nodeToBeDivided.ydim+h,
            nodeToBeDivided.zdim+l, l, w, h, ptsbnw)
60
61      #Divide the node again
62      treeDivision(bnw,maximumPointsAllowed)
63
64      #Find the points that belong to fse
65      ptsfse = []
```

```
66      for point in nodeToBeDivided.getPoints():
67          if point.dimensions[0] >= nodeToBeDivided.xdim+w and point.
                dimensions[0] <= (nodeToBeDivided.xdim+2*w) and point.
                dimensions[1]>=nodeToBeDivided.ydim and point.dimensions
                [1]<=(nodeToBeDivided.ydim+h) and point.dimensions[2]>=
                nodeToBeDivided.zdim and point.dimensions[2]<=(
                nodeToBeDivided.zdim+l):
68              ptsfse.append(point)
69
70      #Create a new node
71      fse = QuadTreeNode(nodeToBeDivided.xdim+w, nodeToBeDivided.ydim,
            nodeToBeDivided.zdim, l, w, h, ptsfse)
72
73      #Divide the node again
74      treeDivision(fse,maximumPointsAllowed)
75
76      #Find the points that belong to bse
77      ptsbse = []
78      for point in nodeToBeDivided.getPoints():
79          if point.dimensions[0] >= nodeToBeDivided.xdim+w and point.
                dimensions[0] <= (nodeToBeDivided.xdim+2*w) and point.
                dimensions[1]>=nodeToBeDivided.ydim and point.dimensions
                [1]<=(nodeToBeDivided.ydim+h) and point.dimensions[2]>=
                nodeToBeDivided.zdim+l and point.dimensions[2]<=(
                nodeToBeDivided.zdim+2*l):
80              ptsbse.append(point)
81
82      #Create a new node
83      bse = QuadTreeNode(nodeToBeDivided.xdim+w, nodeToBeDivided.ydim,
            nodeToBeDivided.zdim+l, l, w, h, ptsbse)
84
85      #Divide the node again
86      treeDivision(bse,maximumPointsAllowed)
87
88      #Find the points that belong to fne
89      ptsfne = []
90      for point in nodeToBeDivided.getPoints():
91          if point.dimensions[0] >= nodeToBeDivided.xdim+w and point.
                dimensions[0] <= (nodeToBeDivided.xdim+2*w) and point.
                dimensions[1]>=nodeToBeDivided.ydim+h and point.dimensions
                [1]<=(nodeToBeDivided.ydim+2*h) and point.dimensions[2]>=
                nodeToBeDivided.zdim and point.dimensions[2]<=(
                nodeToBeDivided.zdim+l):
92              ptsfne.append(point)
93
94      #Create a new node
95      fne = QuadTreeNode(nodeToBeDivided.xdim+w, nodeToBeDivided.ydim+h,
            nodeToBeDivided.zdim, l, w, h, ptsfne)
96
97      #Divide the node again
98      treeDivision(fne,maximumPointsAllowed)
99
100     #Find the points that belong to bne
101     ptsbne = []
102     for point in nodeToBeDivided.getPoints():
103         if point.dimensions[0] >= nodeToBeDivided.xdim+w and point.
                dimensions[0] <= (nodeToBeDivided.xdim+2*w) and point.
                dimensions[1]>=nodeToBeDivided.ydim+h and point.dimensions
                [1]<=(nodeToBeDivided.ydim+2*h) and point.dimensions[2]>=
                nodeToBeDivided.zdim+l and point.dimensions[2]<=(
                nodeToBeDivided.zdim+2*l):
104             ptsbne.append(point)
105
106     #Create a new node
107     bne = QuadTreeNode(nodeToBeDivided.xdim+w, nodeToBeDivided.ydim+h,
            nodeToBeDivided.zdim+l, l, w, h, ptsbne)
108
109     #Divide the node again
110     treeDivision(bne,maximumPointsAllowed)
111
```

```
112        nodeToBeDivided.children = [fsw, bsw, fnw, bnw, fse, bse, fne, bne]
```

Listing 3.4: The Octree Divide Function

Αν και με μία πρώτη ματιά φαίνεται εξαιρετικά πολύπλοκη, η λειτουργία της είναι αρκετά απλή. Αρχικά, αν ο κύβος - κόμβος έχει σημεία λιγότερα από τα μέγιστα επιτρεπόμενα, θεωρείται φύλο και η συνάρτηση τερματίζει (γραμμές 4-7). Στην συνέχεια, εφόσον πρέπει να διαχωριστεί, υπολογίζονται τα μεγέθη των διαστάσεων που θα έχουν τα οκτάντα, τα οποία είναι ίσα με τα μισά του κύβου που χωρίζεται (9-14). Για κάθε ένα από τα 8 οκτάντα που θα δημιουργηθούν εντοπίζονται τα σημεία που πρέπει να περιέχουν και αποθηκεύονται σε μία λίστα. Στην συνέχεια, για κάθε οκτάντα, δημιουργείται ένας νέος κόμβος με σημεία, τα σημεία της λίστας, και η ίδια συνάρτηση καλείται επαναληπτικά χρησιμοποιώντας τον δημιουργημένο κόμβο. Η συνάρτηση θέτει ως παιδιά του κόμβου που την καλεί τα 8 οκτάντα που παράγονται (γραμμή 112). Στο τέλος, έχει δημιουργηθεί στο τελικό δέντρο.

Η δημιουργία του δέντρου με τα σημεία που αντιστοιχούν στα κείμενα της συλλογής γίνεται με το παρακάτω κομμάτι κώδικα:

```
1  from DocumentReprsentationAsPoints import *
2
3  #Get the dataset points from the collection
4  documentVec = DocumentReprsentationAsPoints()
5  print("Finsihed Analyzing the Collection")
6
7  #Find the boundaries
8  minValue = np.min(documentVec)
9  print(minValue)
10 for i in range(len(documentVec)):
11     documentVec[i] = documentVec[i]+ abs(minValue)
12 maxValue = np.max(documentVec)
13
14 #Transform the documentVec into Point objects
15 pointMatrix = []
16
17 for i in range(len(documentVec)):
18     pointMatrix.append(Point(documentVec[i]))
19
20 #Construct the Tree
21 tree_points = pointMatrix[0:1209])
22 print("Constructing the tree")
23 tree = QuadTree(70, tree_points[0:1209], 0, 0, 0, maxValue, maxValue,
       maxValue)
24 print("Dividing the tree")
25 tree.divideTree()
26 print("Finished Constructing and dividing the tree")
```

Listing 3.5: The Creation of the Octree

Σημειώνεται ότι στα σημεία που αντιστοιχούν στα κείμενα και στα ερωτήματα προσθέτεται μία πόλωση, ώστε να μην είναι αρνητικά, γιατί δημιουργούνταν προβλήματα στην σωστή κατασκευή του δέντρου. Επειδή, για την εντοπισμό των κοντινότερων σημείων χρησιμοποιούνται μόνο αποστάσεις από και προς σημεία, αυτή η πόλωση δεν αλλάζει τα αποτελέσματα.

3.2.3 k-Nearest-Neighbors στο Octree

Έχοντας κατασκευάσει το τελικό Octree, χρησιμοποιούνται τα σημεία που αντιστοιχούν στα ερωτήματα της συλλογής για τον εντοπισμό των πιο όμοιων κειμένων. Αυτό γίνεται εντοπίζοντας του κοντινότερους γείτονες για κάθε σημείο ερωτήματος, με την υπόθεση ότι τα κοντινότερα σημεία αντιστοιχούν στα πιο όμοια κείμενα. Η υλοποίηση της συνάρτησης kNearestNeighbors() στο octree φαίνεται στο 3.6.

```
    #k Nearest Neighbors Query
    def kNearestNeighbors(self, point):

        #Set default distance
        r = max(self.root.width, self.root.height, self.root.length)

        #Put root node on a stack
        stack =[]
        stack.append(self.root)

        #Points found
        pointsFound =[]

        #Distance of points
        distPoint =[]

        #Check nodes in respect to r
        while stack:
            nodeToCheck = stack.pop()
            #print(stack)
            for child in nodeToCheck.children:
                #print(len(child.children))
                #print(child.getPoints())
                if intersection(point, r, child):
                    stack.append(child)
                    #print("Pushed to Stack")
                if len(child.children) == 0:
                    for examinedPoint in child.points:
                        #print("Checking point: " + str(examinedPoint))
                        #Get distance
                        pointDim = np.array(point.dimensions)
                        examinedPointDim = np.array(examinedPoint.
                            dimensions)
                        euclDist = np.linalg.norm(pointDim-
                            examinedPointDim)
                        #print(euclDist)
                        #print(r)
                        bisect.insort(distPoint, euclDist)
                        pointIdx = distPoint.index(euclDist)
                        pointsFound.insert(pointIdx, examinedPoint)

                        #Check if distance is less than radius r
                        if euclDist<=r:
                            r = euclDist
                            #print("Point Found")

        #Return closest point
        return pointsFound,distPoint
```

Listing 3.6: kNN query on the Octree

Ο αλγόριθμος που ακολουθήθηκε είναι αυτός που περιγράφεται στις διαφάνειες του μαθήματος. Για τον έλεγχο, αν η σφαίρα γύρω από το σημείο ερωτήματος, τέμνει κάποιον από τους κύβους χρησιμοποιήθηκε ο αλγόριθμος του Jim Arvo και φαίνεται στο 3.7.

```
#Intersection between a ball and a cube
def intersection(center, radius, cube):
    dmin = 0
    dmax = 0
    face = False
    Bmin = [cube.xdim, cube.ydim, cube.zdim]
    Bmax = [cube.xdim+cube.width, cube.ydim+cube.height, cube.zdim+cube
        .length]
    for i in range(3):
        if center.dimensions[i] < Bmin[i]:
            dmin += (center.dimensions[i] - Bmin[i])**2
        elif center.dimensions[i] > Bmax[i]:
            dmin += (center.dimensions[i] - Bmax[i])**2
    if dmin <= radius**2:
        return True

    return False
```

Listing 3.7: Ball and cube intersection

Τέλος, ως πείραμα χρησιμοποιήθηκε ο παρακάτω κώδικας για ένα τυχαίο σημείο ερωτήματος.

```
point_in_map = pointMatrix[1299]

point_found,r = tree.kNearestNeighbors(point_in_map)

print(point_found[1].dimensions)
print(r)

euclDist =[]
for i in range(1209):

    pointDim = np.array(pointMatrix[1299].dimensions)
    euclDist.append(np.linalg.norm(pointDim-documentVec[i]))

euclDist = sorted(euclDist, reverse=False)
print(euclDist)
```

Listing 3.8: Testing kNN on the Octree

3.3 K-d Tree

Το επόμενο προς ανάλυση δέντρο είναι το K-d Tree. Σε σχέση με το Octree η υλοποίηση του K-d Tree ήταν αρκετά πιο απλή αφού για την κατασκευή του χρειάζεται λιγότερη γεωμετρία, μιας και οι διαχωρισμοί γίνονται με ευθείες που τέμνουν τα σημεία κειμένων. Η υλοποίηση του K-d Tree είναι συμβατή και με σημεία n-διαστάσεων, αρκεί να γίνουν μερικές μικρές αλλαγές. Ωστόσο, χρησιμοποιήθηκαν τρισδιάστατα σημεία όπως και στην περίπτωση του Octree. Οι μέθοδοι που περιγράφονται σε αυτό το υπό-κεφάλαιο μπορούν να βρεθούν στο jupyter notebook με όνομα "Kd_tree".

3.3.1 Οι κόμβοι του K-d Tree

Όπως και στην περίπτωση του Octree, θα ξεκινήσουμε αναλύοντας την κλάση που περιγράφει του κόμβους του K-d Tree. Η υλοποίησή της κλάσης αυτής φαίνεται στο 3.9. Κάθε κόμβος του K-d Tree έχει μία μεταβλητή που δηλώνει τον άξονα τον οποίο διαχωρίζει το δέντρο (splittingAxis), μία μεταβλητή με την τιμή που έχει στον άξονα που διαχωρίζει το δέντρο (splittingValue), το σημείο στο οποίο τέμνεται η ευθεία που διαχωρίζει το δέντρο και δύο μεταβλητές που περιέχουν το αριστερό και το δεξιό υποδέντρο, που παράγονται με τον διαχωρισμό.

```
1  #The K-d tree node
2  #splittingAxis is the dimension at which the split happens
3  #siplittingValue is the spliting value
4  #dataElement is the element at the node
5  #leftSub is the left sub tree
6  #rightSub is the right sub tree
7  class K_dTreeNode():
8      def __init__(self, splittingAxis, splittingValue, dataElement,
       leftSub, rightSub):
9          self.splittingAxis = splittingAxis
10         self.splittingValue = splittingValue
11         self.dataElement = dataElement
12         self.leftSub = leftSub
13         self.rightSub = rightSub
```

Listing 3.9: The K-d Node Class

3.3.2 Η κλάση του K-d Tree

Η κύρια κλάση που περιγράφει το K-d Tree. Ως μεταβλητές έχει τα ελάχιστα και τα μέγιστα όρια τιμές στους άξονες και μία μεταβλητή που περιέχει ένα αντικείμενο τύπου K_dTreeNode και αποτελεί την ρίζα του δέντρου. Η υλοποίηση φαίνεται στο 3.10. Η ρίζα παράγεται από μία επαναληπτική διαδικασία, με την οποία το δέντρο χωρίζεται επαναληπτικά.

```
1  #The K_d tree class
2  #startingPoints are the points to be added to the tree
3  #startingX/Y/Z are the x, y, z, starting dimensions
4  #maxX/Y/Z are the maximum x, y, z dimensions on the dataset
5  class K_dTree():
6      def __init__(self,startingPoints, startingX, startingY, startingZ,
       maxX, maxY, maxZ):
7          self.startingX = startingX
8          self.startingY = startingY
9          self.startingZ = startingZ
10         self.maxX = maxX
11         self.maxY = maxY
12         self.maxZ = maxZ
13
14         #Extract the dimensions of the Points
15         pointDimensions = []
16         for point in startingPoints:
17             pointDimensions.append(point.dimensions)
18         self.root_node = splitK_dTree(0,pointDimensions)
```

Listing 3.10: The K-d Tree Class

Η υλοποίηση της συνάρτησης που χωρίζει το δέντρο φαίνεται στο 3.11. Η στρατηγική διάσπασης που ακολουθήθηκε είναι η διάσπαση στο μέσο των τιμών των σημείων του δέντρου. Αρχικά αν δεν υπάρχουν άλλα σημεία για διαχωρισμό η συνάρτηση τερματίζει και ο κόμβος πατέρας που την κάλεσε, δεν έχει είτε δεξιό, είτε αριστερό υποδένδρο. Στην περίπτωση, που δεν έχει κανένα από τα δύο είναι φύλο. Αν έχει περισσευούμενα σημεία τότε αυτά ταξινομούνται σύμφωνα με τον άξονα με τον οποίο γίνεται ο διαχωρισμός. Αρχικά χρησιμοποιείται ως άξονας διάσπασης η πρώτη διάσταση, μετά η δεύτερη, στην συνέχεια η τρίτη, και μετά ξανά η πρώτη. Στην συνέχεια, χωρίζεται ο πίνακας με τα σημεία στο δεξιότερο σημείο με την μέση τιμή, ενημερώνεται ο άξονας διάσπασης και ξανακαλείτε επαναληπτικά η συνάρτηση για τα δύο κομμάτια του πίνακα των σημείων που παράχθηκαν. Τελικά όταν ολοκληρωθεί αυτή η διαδικασία έχει παραχθεί το τελικό K-d Tree.

```
1  def splitK_dTree(splittingAxis, startingPoints):
2
3          #Check if there are no more points
4          if len(startingPoints) == 0:
5              return
6
7          #Divide by finding median
8          #Sort the points according to splittingAxis
9          startingPoints.sort(key=itemgetter(splittingAxis))
10
11         #Find the median towards the end of the list
12         median = int(len(startingPoints)/2)
13         medianData = startingPoints[median]
14         temp =  median + 1
15         for temp in range(len(startingPoints)-1):
16             if startingPoints[temp][splittingAxis] == medianData[
                   splittingAxis]:
17                 median += 1
18                 temp =  median + 1
19         #print(median)
20         #print(len(startingPoints))
21         if median >= len(startingPoints):
22             median = int(median/2)
23         medianData = startingPoints[median]
24
25         #Update the splitting axis
26         newSplittingAxis = (splittingAxis+1) % 3
27
28         #Find left and right points
29         leftSubPoints = startingPoints[0:median]
30         rightSubPoints = startingPoints[median+1:-1]
31
32         #Make a new k_d tree node
33         node = K_dTreeNode(splittingAxis, medianData[splittingAxis],
               medianData, splitK_dTree(newSplittingAxis,leftSubPoints),
               splitK_dTree(newSplittingAxis,rightSubPoints))
34
35         return node
```

Listing 3.11: The K-d Splitting Function

Η δημιουργία του δέντρου με τα σημεία που αντιστοιχούν στα κείμενα της συλλογής γίνεται με το παρακάτω κομμάτι κώδικα:

```
1  from DocumentReprsentationAsPoints import *
2
3  #Get the dataset points from the collection
4  documentVec = DocumentReprsentationAsPoints()
5  print("Finsihed Analyzing the Collection")
6
7  #Find the boundaries
8  minValue = np.min(documentVec)
9
10 for i in range(len(documentVec)):
11     documentVec[i] = documentVec[i]+ abs(minValue)
12 maxValue = np.max(documentVec)
13
14 #Transform the documentVec into Point objects
15 pointMatrix = []
16
17 for i in range(len(documentVec)):
18     pointMatrix.append(Point(documentVec[i]))
19
20 #Construct the Tree
21 print("Constructing the tree")
22 tree = K_dTree(pointMatrix[0:1209], 0, 0, 0, maxValue, maxValue,
       maxValue)
23 print("Finished Constructing and dividing the tree")
```

Listing 3.12: The Creation of the K-d Tree

Σημειώνεται ότι στα σημεία που αντιστοιχούν στα κείμενα και στα ερωτήματα προσθέτεται μία πόλωση, ώστε να μην είναι αρνητικά, γιατί δημιουργούνταν προβλήματα στην σωστή κατασκευή του δέντρου. Επειδή, για την εντοπισμό των κοντινότερων σημείων χρησιμοποιούνται μόνο αποστάσεις από και προς σημεία, αυτή η πόλωση δεν αλλάζει τα αποτελέσματα.

3.3.3 k-Nearest-Neighbors στο K-d Tree

Ολοκληρώνοντας την κατασκευή του K-d Tree είμαστε σε θέση να υλοποιήσουμε μία συνάρτηση για ερωτήματα τύπου kNN στο δέντρο. Η υλοποίηση της συνάρτησης φαίνεται στο 3.13. Αρχικά, θέτεται η μέγιστη απόσταση r.

```
1      #kNearestNeighbors Query
2      def kNearestNeighbors(self, point):
3
4          #Set default distance
5          r = max(self.maxX, self.maxY, self.maxZ)
6
7          #The distance Matrix
8          distPoint = []
9
10         #The Point Matrix
11         pointsFound = []
12
13
14         pointsFound, distPoint, r = self.root_node.traverseTree(point,
               r, distPoint, pointsFound)
15
16         return pointsFound, distPoint
```

Listing 3.13: kNN query on the K-d Tree

Το ενδιαφέρον δεν είναι στην συγκεκριμένη συνάρτηση που είναι υπερβολικά απλή αλλά στην συνάρτηση traverseTree που καλεί. Η υλοποίηση της συνάρτησης traverseTree φαίνεται στο 3.14. Αν και φαίνεται πολύπλοκη, η λειτουργία της γίνεται εύκολα κατανοητή. Αρχικά, για το δοθέν σημείο διασχίζεται το δέντρο, ώσπου να βρεθεί ο κόμβος φύλο στον οποίο ανήκει και υπολογίζεται η απόσταση από το σημείο του κόμβου, ανανεώνοντας την μέγιστη απόσταση r. Σε αυτό το σημείο ο αλγόριθμος επισκέπτεται επαναληπτικά τα σημεία που πέρασε για να φτάσει στο κόμβο φύλο και υπολογίζει την απόσταση από το σημείο κάθε κόμβου, ανανεώνοντας την μέγιστη απόσταση r, σε περίπτωση που βρεθεί νέα ελάχιστη απόσταση. Σε περίπτωση που υπάρχει για ένα κόμβο και το άλλο υποδένδρο και η μέγιστη απόσταση r είναι μεγαλύτερη από την απόσταση στο άλλο υποδένδρο, ο αλγόριθμος επισκέπτεται και το άλλο υποδένδρο.

```
1  def traverseTree(self, point, r , distPoint, pointsFound):
2          #Get Examined node Axis values
3          examinedNodeSplittingAxis = self.splittingAxis
4          #print(examinedNodeSplittingAxis)
5          examinedNodeSplittingValue = self.splittingValue
6          #print(examinedNodeSplittingValue)
7
8          #Debugging Checks
9          #print(self.leftSub)
10         #print(self.rightSub)
11         #print(not bool(self.leftSub))
12         #print(not bool(self.rightSub))
13         #print(not bool(self.leftSub) and bool(self.rightSub))
14
15         #Traverse the tree until we reach a leaf and not bool(self.
               rightSub):
16         if point.dimensions[examinedNodeSplittingAxis]<=
               examinedNodeSplittingValue:
17             if not bool(self.leftSub):
18                 #This is a leaf node
19                 print("Reached a leaf node")
20
21                 #Calculate current nodes distance from point and if it
                       is better then update r
22                 pointDim = np.array(point.dimensions)
23                 examinedPointDim = np.array(self.dataElement)
24                 euclDist = np.linalg.norm(pointDim-examinedPointDim)
25
26                 bisect.insort(distPoint, euclDist)
27                 pointIdx = distPoint.index(euclDist)
28                 pointsFound.insert(pointIdx, self)
29
30                 #Check if distance is less than radius r
31                 if euclDist<=r:
32                     r = euclDist
33                     #print("Point Found")
34
35                 return pointsFound, distPoint, r
36
37             else:
38                 pointsFound, distPoint, r = self.leftSub.traverseTree(
                       point, r,  distPoint, pointsFound)
39                 print("Traversing the left subtree")
40
41                 #Calculate current nodes distance from point and if it
                       is better then update r
42                 pointDim = np.array(point.dimensions)
43                 examinedPointDim = np.array(self.dataElement)
44                 euclDist = np.linalg.norm(pointDim-examinedPointDim)
```

```
45
46                  bisect.insort(distPoint, euclDist)
47                  pointIdx = distPoint.index(euclDist)
48                  pointsFound.insert(pointIdx, self)
49
50                  #Check if distance is less than radius r
51                  if euclDist<=r:
52                      r = euclDist
53                      #print("Point Found")
54
55
56                  #If there is a right side
57                  if bool(self.rightSub):
58
59                      #Calculate the distance between the current node
                            and the right side node
60                      nodeDist = np.linalg.norm(self.splittingValue-self.
                            rightSub.splittingValue)
61
62                      #If current best is bigger than the distance then
                            traverse the right side too
63                      if r>nodeDist:
64                          self.rightSub.traverseTree(point, r, distPoint,
                                pointsFound)
65
66                  return pointsFound, distPoint, r
67
68          else:
69              if not bool(self.rightSub):
70                  #This is a leaf node
71                  print("Reached a leaf node")
72
73                  #Calculate current nodes distance from point and if it
                        is better then update r
74                  pointDim = np.array(point.dimensions)
75                  examinedPointDim = np.array(self.dataElement)
76                  euclDist = np.linalg.norm(pointDim-examinedPointDim)
77
78                  bisect.insort(distPoint, euclDist)
79                  pointIdx = distPoint.index(euclDist)
80                  pointsFound.insert(pointIdx, self)
81
82                  #Check if distance is less than radius r
83                  if euclDist<=r:
84                      r = euclDist
85                      #print("Point Found")
86
87                  return pointsFound, distPoint, r
88
89              else:
90                  pointsFound, distPoint, r = self.rightSub.traverseTree(
                        point, r, distPoint, pointsFound)
91                  print("Traversing the rightSub subtree")
92
93                  #Calculate current nodes distance from point and if it
                        is better then update r
94                  pointDim = np.array(point.dimensions)
95                  examinedPointDim = np.array(self.dataElement)
96                  euclDist = np.linalg.norm(pointDim-examinedPointDim)
97
98                  bisect.insort(distPoint, euclDist)
99                  pointIdx = distPoint.index(euclDist)
100                 pointsFound.insert(pointIdx, self)
101
102                 #Check if distance is less than radius r
103                 if euclDist<=r:
104                     r = euclDist
105                     #print("Point Found")
106
107                 #If there is a left side
108                 if bool(self.leftSub):
109
110                     #Calculate the distance between the current node
                            and left right side node
```

```
111                     nodeDist = np.linalg.norm(self.splittingValue-self.
                            leftSub.splittingValue)
112
113                     #If current best is bigger than the distance then
                            traverse the right side too
114                     if r>nodeDist:
115                         self.leftSub.traverseTree(point, r, distPoint,
                                pointsFound)
116
117                 return pointsFound, distPoint, r
```

Listing 3.14: The traverseTree function

Τέλος, ως πείραμα χρησιμοποιήθηκε ο παρακάτω κώδικας για ένα τυχαίο σημείο ερωτήματος.

```
1  point_in_map = pointMatrix[1299]
2
3  points, distances = tree.kNearestNeighbors(point_in_map)
4
5  print(points[-1])
6  print(distances)
7
8  euclDist =[]
9  for i in range(1209):
10
11      pointDim = np.array(pointMatrix[1299].dimensions)
12      euclDist.append(np.linalg.norm(pointDim-documentVec[i]))
13
14 euclDist = sorted(euclDist, reverse=False)
15 print(euclDist)
```

Listing 3.15: Testing kNN on the K-d Tree

3.4 R-tree

Το τελευταίο δέντρο που υλοποιήθηκε είναι το R-tree. Το R-tree χρησιμοποιεί Minimum Bounding Rectangles (MBR), στην περίπτωση μας Minimum Bounding Cubes. Η λειτουργία του θυμίζει λίγο το Octree με την κύρια διαφορά ότι αντί octants, το R-tree χρησιμοποιεί MBR. Οι μέθοδοι που περιγράφονται σε αυτό το υπό-κεφάλαιο μπορούν να βρεθούν στο jupyter notebook με όνομα "R-tree".

3.4.1 Οι κόμβοι του R-tree

Όπως και στις προηγούμενες περιπτώσεις αρχικά αναλύεται η κλάση που περιγράφει του κόμβους του R-tree. Η υλοποίηση της φαίνεται στο 3.16. Κάθε κόμβος του R-tree περιγράφεται από ένα MBR. Αυτό περιέχει άλλα MBRs ή περιέχει και τέμνει σημεία αν είναι φύλο. Στην μεταβλητή objectPtr, περιέχεται μία λίστα με άλλους κόμβους ή σημεία αν είναι φύλο. Στην μεταβλητή parentNode αποθηκεύετε το αντικείμενο που περιγράφει τον κόμβο που περιέχει το παρόν κόμβο. Τέλος, υπάρχει μία μεταβλητή σημαία που δηλώνει αν ο παρόν κόμβος είναι φύλο ή όχι.

```
#class for the nodes of the r-tree
#nodeMBR is the MBR that describes the node
#objectPtr points to the data in the node --> either another node or
    points
#isLeaf determes the type of data
#parentNode points to the parent node
class rTreeNode():
    def __init__(self,x_low, x_high, y_low, y_high, z_low, z_high, data
        , leafFlag, parentNode):
        self.nodeMBR = MBR(x_low, x_high, y_low, y_high, z_low, z_high)
        self.objectPtr = data
        self.isLeaf = leafFlag
        self.parentNode = parentNode
```

Listing 3.16: The R-tree node class

Η κλάση για το MBR φαίνεται στο 3.17. Οι μεταβλητές περιγράφουν όλες τις γωνίες του κύβου.

```
#Class for the minimum bounding cube
#x/y/z low is the lowest x/y/z coordinate
#x/y/z high is the highest x/y/z coordinate
class MBR():
    def __init__(self,x_low, x_high, y_low, y_high, z_low, z_high):
        self.x_low = x_low
        self.x_high = x_high
        self.y_low = y_low
        self.y_high = y_high
        self.z_low = z_low
        self.z_high = z_high
```

Listing 3.17: The R-tree mbr class

3.4.2 Η κλάση του R-tree

Σε αυτό το υπό-κεφάλαιο περιγράφεται η κύρια κλάση του R-tree. Η υλοποίηση της κλάσης φαίνεται στο 3.18. Το R-tree έχει δύο μεταβλητές m και M που αντιστοιχούν στον ελάχιστο και μέγιστο αριθμό σημείων που μπορεί να περιέχει ένας κόμβος.

```
#class for the r-tree
class rTree():
    def __init__(self, m, M, data):
        self.mEntries = m
        self.MEntries = M
        self.rootNode = rTreeConstruction(m, M, data)
```

Listing 3.18: The R-tree class

Επίσης έχει μία μεταβλητή, με όνομα rootNode, που αποτελεί την ρίζα του δέντρου και είναι ένα αντικείμενο τύπου rTreeNode. Κατά την δημιουργία του δένδρου καλείται η συνάρτηση rTreeConstruction(), της οποίας η υλοποίηση φαίνεται στο 3.19.

```
#Constructs the r-tree
#data is our points
def rTreeConstruction(m, M, data):

    #Extract the dimensions of the Points
```

```
6       pointDimensions = []
7
8       for point in data:
9           pointDimensions.append(point.dimensions)
10
11      pointDimensions = numpy.array(pointDimensions)
12
13      #Find the maximum and minimum x/y/z
14      x_low, x_high, y_low, y_high, z_low, z_high = findBoundaries(
            pointDimensions)
15
16      rootNode = rTreeNode(x_low, x_high, y_low, y_high, z_low, z_high,
            splitRTree(None,m, M, pointDimensions), False, None)
17
18      return rootNode
```

Listing 3.19: The R-tree construction function

Η συνάρτηση αρχικά εντοπίζει τα μέγιστα όρια που θα έχει το αρχικό MBR. Αυτό το επιτυγχάνει με την βοήθεια της συνάρτησης findBoundaries(), της οποίας η υλοποίηση φαίνεται στο 3.20. Η συνάρτηση findBoundaries διατρέχει την λίστα με τα σημεία και βρίσκει τις ελάχιστες και μέγιστες τιμές για κάθε διάσταση.

```
1   #Find Boundaries of points
2   def findBoundaries(pointDimensions):
3       #print(pointDimensions)
4
5       #Find the maximum and minimum x/y/z
6       x_low = pointDimensions[0][0]
7       x_high = pointDimensions[0][0]
8       y_low = pointDimensions[0][1]
9       y_high = pointDimensions[0][1]
10      z_low = pointDimensions[0][2]
11      z_high = pointDimensions[0][2]
12
13      for x in range(1,len(pointDimensions)):
14
15          #x dimension
16          if pointDimensions[x][0] < x_low:
17              x_low = pointDimensions[x][0]
18          elif pointDimensions[x][0] > x_high:
19              x_high = pointDimensions[x][0]
20
21          #y dimension
22          if pointDimensions[x][1] < y_low:
23              y_low = pointDimensions[x][1]
24          elif pointDimensions[x][1] > y_high:
25              y_high = pointDimensions[x][1]
26
27          #z dimesion
28          if pointDimensions[x][2] < z_low:
29              z_low = pointDimensions[x][2]
30          elif pointDimensions[x][2] > z_high:
31              z_high = pointDimensions[x][2]
32      return x_low, x_high, y_low, y_high, z_low, z_high
```

Listing 3.20: The findBoundaries function

Στην συνέχεια, δημιουργείται ο πρώτος κόμβος, ενώ ταυτόχρονα καλείται η συνάρτηση splitRTree(), η οποία αναλαμβάνει την διάσπαση του δέντρου σε κόμβους. Η υλοποίηση της συνάρτησης φαίνεται στο 3.21. Η splitRTree() χωρίζει αναδρομικά το δέντρο σε κόμβους. Αρχικά, ορίζει το περίγραμμα που σχηματίζεται από τα σημεία (Convex Hull) και εντοπίζει από τα σημεία που τον απαρτίζουν, τα δύο με την μέγιστη απόσ-

ταση (seeds). Στην συνέχεια, κάθε ένα από τα υπολειπόμενα σημεία αναθέτετε στο seed, με το οποίο απέχει την λιγότερη απόσταση. Στην περίπτωση, που ένα από τα δύο seed περιέχει λιγότερα από m σημεία, γίνεται μεταφορά των κοντινότερων σε αυτό σημείων από το άλλο seed. Τέλος, αν κάποιο seed έχει περισσότερα από M σημεία τότε δημιουργείται για αυτό το seed ένας κόμβος και συνεχίζει η διάσπαση σε αυτόν, ειδάλλως δημιουργείται ένα κόμβος φύλο. Όταν ολοκληρωθεί η διαδικασία έχει δημιουργηθεί το τελικό R-tree.

```
1  #Splits the r-tree
2  def splitRTree(self,m, M, pointDimensions):
3
4      #Partition the point matrix into 2 groups
5      #First find the two seed points using the convex hull of the data
6      #Extract the points forming the hull
7      try:
8          convexHull = ConvexHull(pointDimensions)
9          convexHullPoints = pointDimensions[convexHull.vertices,:]
10     except Exception:
11         convexHullPoints = pointDimensions
12     #print(convexHull.vertices)
13     #print(pointDimensions)
14     #print(pointDimensions[convexHull.vertices,:])
15
16     #Find the distance naively using the convex hull points
17     maxDist = 0
18     pointA = pointDimensions[0]
19     pointB = pointDimensions[-1]
20     for x in range(len(convexHullPoints)):
21         for y in range(len(convexHullPoints)):
22             euclDist = numpy.linalg.norm(convexHullPoints[x]-
                   convexHullPoints[y])
23             if euclDist>maxDist:
24                 pointA = convexHullPoints[x]
25                 pointB = convexHullPoints[y]
26                 maxDist = euclDist
27
28     #assign each point in one group
29     A_group_points = []
30     B_group_points = []
31     for i in range(len(pointDimensions)):
32         euclDistA = numpy.linalg.norm(pointA-pointDimensions[i])
33         euclDistB = numpy.linalg.norm(pointB-pointDimensions[i])
34         if euclDistA>euclDistB:
35             B_group_points.append(pointDimensions[i])
36         else:
37             A_group_points.append(pointDimensions[i])
38
39
40     #Check if both groups have m elements and if they dont balance them
41     if len(A_group_points)<m:
42         dist = []
43         for i in range(len(B_group_points)):
44             dist.append([i,numpy.linalg.norm(pointA-B_group_points[i])
                   ])
45         diff = m- len(A_group_points)
46         dist.sort(key=itemgetter(0))
47         for y in range(diff):
48             A_group_points.append(B_group_points[dist[y][0]])
49             del B_group_points[dist[y][0]]
50
51     if len(B_group_points)<m:
52         dist = []
53         for i in range(len(A_group_points)):
54             dist.append([i,numpy.linalg.norm(pointB-A_group_points[i])
```

```
                ])
55          diff = m - len(B_group_points)
56          dist.sort(key=itemgetter(0))
57          #print(dist[1][1])
58          for y in range(diff):
59              B_group_points.append(A_group_points[dist[y][0]])
60              del A_group_points[dist[y][0]]
61
62      #print(len(A_group_points))
63      #print(len(B_group_points))
64      A_group_points = numpy.array(A_group_points)
65      B_group_points = numpy.array(B_group_points)
66      #Find the boudaries for each group
67      boundA_x_low, boundA_x_high, boundA_y_low, boundA_y_high,
            boundA_z_low, boundA_z_high = findBoundaries(A_group_points)
68      boundB_x_low, boundB_x_high, boundB_y_low, boundB_y_high,
            boundB_z_low, boundB_z_high = findBoundaries(B_group_points)
69
70      #Split the tree recursively
71      if len(A_group_points)>M:
72          print("Still Spliting")
73          nodeA = rTreeNode(boundA_x_low, boundA_x_high, boundA_y_low,
                boundA_y_high, boundA_z_low, boundA_z_high, splitRTree(self
                ,m, M, A_group_points), False, self)
74      else:
75          print("Created a leaf")
76          nodeA = rTreeNode(boundA_x_low, boundA_x_high, boundA_y_low,
                boundA_y_high, boundA_z_low, boundA_z_high, A_group_points,
                 True, self)
77      if len(B_group_points)>M:
78          print("Still Spliting")
79          nodeB = rTreeNode(boundB_x_low, boundB_x_high, boundB_y_low,
                boundB_y_high, boundB_z_low, boundB_z_high, splitRTree(self
                ,m, M, B_group_points), False, self)
80      else:
81          print("Created a leaf")
82          nodeB = rTreeNode(boundB_x_low, boundB_x_high, boundB_y_low,
                boundB_y_high, boundB_z_low, boundB_z_high, B_group_points,
                 True, self)
83
84      return [nodeA, nodeB]
```

Listing 3.21: The splitRTree function

Η δημιουργία του δέντρου με τα σημεία που αντιστοιχούν στα κείμενα της συλλογής γίνεται με το παρακάτω κομμάτι κώδικα:

```
1  from DocumentReprsentationAsPoints import *
2
3  #Get the dataset points from the collection
4  documentVec = DocumentReprsentationAsPoints()
5  print("Finsihed Analyzing the Collection")
6
7  #Find the boundaries
8  minValue = numpy.min(documentVec)
9  for i in range(len(documentVec)):
10     documentVec[i] = documentVec[i]+ abs(minValue)
11 maxValue = numpy.max(documentVec)
12
13 #Transform the documentVec into Point objects
14 pointMatrix = []
15
16 for i in range(len(documentVec)):
17     pointMatrix.append(Point(documentVec[i]))
18
19 #Construct the Tree
20 print("Constructing the tree")
21 tree = rTree(25,50, pointMatrix)
```

```
22 print("Finished Constructing and spliting the tree")
```

Listing 3.22: The Creation of the R-tree

Σημειώνεται ότι στα σημεία που αντιστοιχούν στα κείμενα και στα ερωτήματα προσθέτεται μία πόλωση, ώστε να μην είναι αρνητικά, γιατί δημιουργούνταν προβλήματα στην σωστή κατασκευή του δέντρου. Επειδή, για την εντοπισμό των κοντινότερων σημείων χρησιμοποιούνται μόνο αποστάσεις από και προς σημεία, αυτή η πόλωση δεν αλλάζει τα αποτελέσματα.

3.4.3 k-Nearest-Neighbors στο R-tree

Η διαδικασία που ακολουθείται για τα ερωτήματα kNN στο R-tree είναι η ίδια με αυτή στο Octree. Τέλος, ως πείραμα για την ορθή λειτουργία των ερωτημάτων k-NN στο R-tree χρησιμοποιήθηκε ο παρακάτω κώδικας για ένα τυχαίο σημείο ερωτήματος.

```
1  point_in_map = pointMatrix[1299]
2  print("Checking Point:" + str(point_in_map.dimensions))
3
4  point_found,r = tree.kNearestNeighbors(point_in_map,100)
5
6  print(point_found[1])
7  print(r)
8
9  euclDist =[]
10 for i in range(1209):
11
12     pointDim = numpy.array(pointMatrix[1299].dimensions)
13     euclDist.append(numpy.linalg.norm(pointDim-documentVec[i]))
14
15 euclDist = sorted(euclDist, reverse=False)
16 print("Testing List:")
17 print(euclDist)
```

Listing 3.23: Testing kNN on the R-tree

Chapter 4

Επίλογος

Όπως έχει ήδη αναφερθεί μετά από συζήτηση με τον υπεύθυνο καθηγητή κ. Σιούτα Σπυρίδων, δόθηκε η επιλογή μη χρήσης της μεθόδου LSH, ακόμα και αν ζητείται από την εκφώνηση. Το παρόν μάθημα αποτελεί το τελευταίο μου μάθημα για πτυχίο και για το λόγο αυτό θα παρακαλούσα για την όσο το δυνατό επιείκεια σας. Σας ευχαριστώ πολύ.

Βιβλιογραφία

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, 487–499. http://dl.acm.org/citation.cfm?id=645920.672836

Baeza-Yates, R., & Ribeiro-Neto, B. (2008). *Modern information retrieval: The concepts and technology behind search* (2nd ed.). Addison-Wesley Publishing Company.

Maćkiewicz, A., & Ratajczak, W. (1993). Principal components analysis (pca). *Computers Geosciences*, *19*(3), 303–342. https://doi.org/https://doi.org/10.1016/0098-3004(93)90090-R

Possas, B., Ziviani, N., Meira, W., Jr., & Ribeiro-Neto, B. (2002). Set-based model: A new approach for information retrieval. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 230–237. https://doi.org/10.1145/564376.564417

Shaw, M., W., Wood, B., J., Wood, B., J., & Tibbo, R., H. (1991). The cystic fibrosis database: Content and research opportunities.

Steven Bird, E. K., & Loper, E. (2009). *Natural language processing with python.* https://www.nltk.org/book/ch02.html

Listings

2.1 Representing the documents and queries as points 3
2.2 Analyzing the documents and queries 5
2.3 The function that generates the one termsets 6
2.4 The apriori algorithm 6
2.5 Termset frequency calculation 7
2.6 Termset document frequency calculation 8
2.7 $TF \times IDF$ calculation 8
3.1 Point Class 10
3.2 The Octree Node Class 11
3.3 The Octree Class 12
3.4 The Octree Divide Function 12
3.5 The Creation of the Octree 15
3.6 kNN query on the Octree 16
3.7 Ball and cube intersection 17
3.8 Testing kNN on the Octree 17
3.9 The K-d Node Class 18
3.10 The K-d Tree Class 18
3.11 The K-d Splitting Function 19
3.12 The Creation of the K-d Tree 20
3.13 kNN query on the K-d Tree 20
3.14 The traverseTree function 21
3.15 Testing kNN on the K-d Tree 23
3.16 The R-tree node class 24
3.17 The R-tree mbr class 24
3.18 The R-tree class 24
3.19 The R-tree construction function 24
3.20 The findBoundaries function 25
3.21 The splitRTree function 26
3.22 The Creation of the R-tree 27
3.23 Testing kNN on the R-tree 28