Αλγόριθμοι για Δεδομένα Ευρείας Κλίμακας

LAB1 REPORT:
Κουφόπουλος Μύρων : 4398
Χριστόπουλος Κωνσταντίνος : 4527

Στην αναφορά μας θα αναλύσουμε την λογική που ακολουθήσαμε και συμπεράσματα-παρατηρήσεις μας για κάθε βήμα. Στο κώδικά υπάρχουν αναλυτικά σχόλια για επεξήγηση της υλοποίηση μας.

Βήμα 1 : Ανάγνωση και Επεξεργασία Δεδομένων Εισόδου

Στο πρώτο βήμα της υλοποίησης δημιουργήσαμε μια δομή δεδομένων στην μνήμη η οποία αποθηκεύει μόνο τα πραγματικά σύνολα λέξεων (ως συλλογές από διαφορετικές wordID τιμές).

Για την συνάρτηση αυτή ξεκινάμε να διαβάζουμε από την 4 γραμμή (όπου βρίσκετε το έγγραφο #1 της συλλογής) του εκάστοτε αρχείου που έχουμε διότι από εκεί αρχίζει η πληροφορία που χρειαζόμαστε. Διαβάζουμε γραμμή-γραμμή το αρχείο για τις πρώτες numDocuments εγγραφές και για κάθε ξεχωριστό docID δημιουργούμε ένα διαφορετικό frozenset που περιέχει τα wordIDs του. Στη συνέχεια εισάγουμε όλα τα frozenset σε μία λίστα (wordSet).

```
9  def MyReadDataRoutine(dataDocument, numDocuments):
10     #diavazw to arxeio mexri ta prwta numDocuments
11     #kataskeuazw frozensets
12     next(f)          # gia na pame grammh 3 kai me t
13     n = 1            # se poio dicID briskomaste na
14     A = []           # h lista pou pairnei ta wordID
15     wordSet = []     # h lista me ta frozensets (sum
16     while (n <= numDocuments):
17         x = f.readline().rstrip()
18         x = x.split()
19         if (int(x[0]) >= n):        # gia na allaze
20             n += 1
21             B = tuple(A)
22             fSet = frozenset(B)
23             wordSet.append(fSet)
24             A = []
25             if (n > numDocuments):
26                 break
27         A.append(int(x[1]))
28     return wordSet
```

Βήμα 2 : Συναρτήσεις για Jaccard Ομοιότητα

Σε αυτό το βήμα γνωρίζοντας το τύπο της Jaccard Similarity. Παρατηρούμε πως ο μόνος άγνωστος είναι η τιμή της τομής των δύο συνόλων και παρακάτω την υπολογίζουμε με δύο διαφορετικούς τρόπους.

$$JacSim(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

1) Απλός τρόπος υπολογισμού της τομής :

Σε αυτό το σημείο συγκρίνουμε απευθείας δύο δομές τύπου frozenset δηλαδή συγκρινούμε τα wordIDs δύο διαφορετικων docIDs. Υπολογίζουμε την τιμή με διπλό for-loop γιατί το κάθε frozenset δεν περιέχει τα στοιχεία του σε αύξουσα σειρά.

```python
def MyJacSimWithSets(docID1,docID2):
    #docID1.intersection(docID2) for fun!
    counterIntersection = 0
    for wordID1 in docID1:
        for wordID2 in docID2:
            if (wordID1 == wordID2):
                counterIntersection += 1
    jacSim = counterIntersection / (len(docID1) + len(docID2) - counterIntersection)
    return jacSim

```

2) Αποδοτικός τρόπος υπολογισμού της τομής :

Ένας πιο αποδοτικός τρόπος υπολογισμού της τομής είναι να εκμεταλλευτούμε το γεγονός πως δεν έχουμε διπλότυπα στα frozenset μας. Δίνουμε ως όρισμα στη συνάρτησή μας δύο ταξινομημένες λίστες που αντιστοιχούν στα 2 εκάστοτε docID που θέλουμε να υπολογίσουμε την jaccard similarity τους. Πιο συγκεκριμένα, με έναν δείκτη στη κάθε λίστα μπορούμε σε σχεδόν γραμμικό χρόνο να υπολογίσουμε την τομή τους.

```python
def MyJacSimWithOrderedLists(docID1, docID2):
    pos1 = 0
    pos2 = 0
    counterIntersection = 0
    len1 = len(docID1)
    len2 = len(docID2)
    while (pos1 < len1 and pos2 < len2):
        if (docID1[pos1] == docID2[pos2]):
            counterIntersection += 1
            pos1 += 1
            pos2 += 1
        else:
            if docID1[pos1] < docID2[pos2]:
                pos1 += 1
            else:
                pos2 += 1
    jacSim = counterIntersection / (len(docID1) + len(docID2) - counterIntersection)
    return jacSim
```

Παρακάτω παρατίθεται ένα screenshot για την ομοιότητα των docID (εγγράφων) 779 και 1073 που υπάρχουν στο αρχείο DATA_1-docword.enron.txt. Στην τελική υλοποίηση μας δεν υπάρχει ο counter comparisons χρησιμοποιήθηκε μόνο για να παρατηρήσουμε την μεγάλη διαφορά συγκρίσεων.

```
================================================================
time collapsed with sets: 0.017007
JacSim2loop = 0.045128
#Comparisons =  168688
================================================================
================================================================
time collapsed with ordered lists : 0.000000
jacSimOrdered = 0.045128
#Comparisons =  974
================================================================
```

Βήμα 3 : Συναρτήσεις για Ομοιότητα Υπογραφών

Δημιουργούμε μια τυχαία συνάρτηση κατακερματισμού η οποία μας βοηθάει να ελαχιστοποιήσουμε την πιθανότητα 2 διαφορετικά wordIDs να βρεθούν στον ίδιο κάδο. Μπορούν να δημιουργηθούν αρκετοί κάδοι ώστε να εξασφαλίζετε ότι κάθε φορά που την καλούμε είναι απίθανο δύο

διαφορετικοί αριθμοί να πέσουν στον ίδιο κάδο εκτός και αν είναι ίδιοι. Θα την χρησιμοποιήσουμε παρακάτω για να έχουμε τυχαίες μεταθέσεις και στον LSH.

```
60  def create_random_hash_function(p=2**33-355, m=2**32-1):
61      a = random.randint(1,p-1)
62      b = random.randint(0,p-1)
63      return lambda x: 1 + (((a * x + b) %p) %m)
```

Στη συνάρτηση αυτή δημιουργούμε-διαβάζουμε ένα αρχείο στο οποίο αποθηκεύουμε τον πίνακα μεταθέσεων(μορφή : key-value) που θα χρειαστεί για το min-hashing. Συγκεκριμένα θα ακολουθήσει την try αν το αρχείο μας θέλουμε είτε να δημιουργηθεί για πρώτη φορά είτε να ξαναδημιουργηθεί. Φτιάχνουμε ένα λεξικό για κάθε ζεύγος W κλειδιού-τιμής και το ταξινομούμε με βάση την τιμή του κάθε κλειδιού η οποία είναι τυχαία. Έπειτα αντικαθιστούμε την τιμή κάθε ζεύγους με την θέση του ζεύγους αυτού μέσα στο λεξικό. Τέλος αποθηκεύουμε το κάθε λεξικό το οποίο αναπαριστά κάθε μετάθεση σε μια λίστα(permutations). Αν το αρχείο προ-υπάρχει τότε το ανοίγουμε, το διαβάζουμε και αποθηκεύουμε τα δεδομένα του στην λίστα permutations.

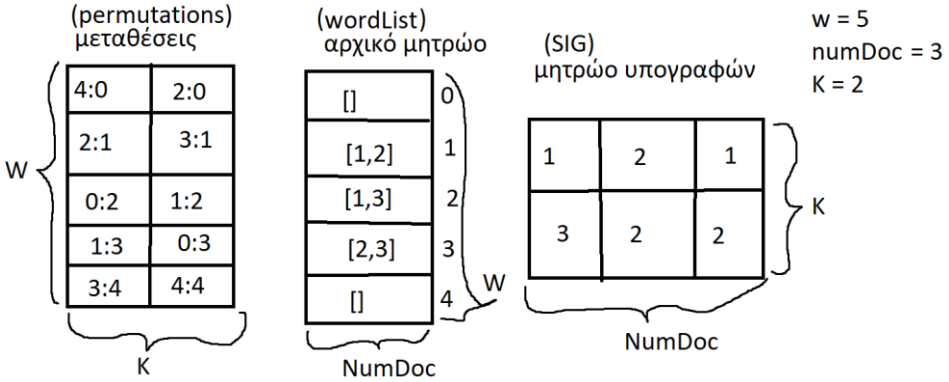
```
69  def hashFunction(K,W):
70      # mporousame na xrhsimopoihsoume json arxeia gia pio eukola dn to eidame nwris
71      permutations = []
72      try:
73          filename = 'hashFunctions.csv'
74          with open(filename, 'x', newline="") as f:
75              csvwriter = csv.writer(f)              # 2. create a csvwriter object
76              csvwriter.writerow(['key', 'value']) # 4. write the header
77              for i in range(K):
78                  h = create_random_hash_function()
79                  randomHash = {x:h(x) for x in range(int(W))}
80                  myHashKeysOrderedByValues = sorted(randomHash, key=randomHash.get)
81                  myHash = {myHashKeysOrderedByValues[x]:x for x in range(int(W)) }
82                  permutations.append(myHash)      # pinakas
83                  myListHash = myHash.items()
84                  csvwriter.writerows(myListHash) # 5. write the rest of the data
85              f.close()
86      # an uparxei hdh to arxeio den theloume na to ksanadhmiourghsei
87      except FileExistsError:
88          with open('hashFunctions.csv', mode='r') as infile:
89              reader = csv.reader(infile)
90              next(reader)
91              wStart = int(W) + 1                 # epeidh xekiname apo thn 2h gramm
92              temp1 = list(reader)                # diabazei olo to csv arxeio kai t
93              tempInt = []
94              for x in temp1:
95                  x = str_list_to_int_list(x)
96                  tempInt.append(x)
97              n = 0                               # counter gia na allazoume ta dioc
98              for k in range(K):
99                  temp = tempInt[n:int(W)+n]      # kratame se lista posa stoixeia e
100                 temp_dict = dict(temp)          # metatrepoume thn lista se dictio
101                 permutations.append(temp_dict)  # bazoume to ekastote dictionary s
102                 n += int(W)                     # gia na diabasei ta epomena W sto
103     return permutations
```

Πρώτα αρχικοποιούμε τις λίστες που θα χρειαστούμε. Στο SIG θα αποθηκευτεί το αποτέλεσμα του minhash δηλαδή το μητρώο υπογραφών. Το μέγεθος του διαμορφώνουμε ως έναν δισδιάστατο πίνακα όπου οι γραμμές θα είναι όσες το πλήθος των μεταθέσεων και οι στήλες θα είναι όσες τα numDocuments(len(wordset)). Αρχικοποιούμε κάθε θέση του στο άπειρο ώστε οποιαδήποτε και αν είναι η θέση γραμμής στη μετάθεση να αντικαταστήσει την τιμή αυτή. Το wordList θα αναπαριστά το αρχικό μας μητρώο, αρχικοποιείται ως ένα δισδιάστατο πίνακας με μέγεθος W όσες θα είναι και οι διαφορετικές λέξεις που θα έχουμε. Στο wordLists θα αποθηκευτούν τα δεδομένα των frozensets ταξινομημένα για αυτό αρχικοποιείται στο πλήθος τους(numDocuments). Συνεχίζοντας, δημιουργούμε το πίνακα αρχικού μητρώου ο οποίος δεν θα αναπαριστάται από 0/ή1 αλλά κάθε γραμμή του (αντιστοιχεί σε wordID-1, επειδή ξεκινάει η αρίθμηση από το 0) θα περιέχει τα docIDs τα

οποία εμφανίζουν το εκάστοτε wordID. Στις γραμμές 127-135 διατρέχουμε γραμμή-γραμμή το πίνακα αρχικού μητρώου. Όσες γραμμές είναι κενές τις προσπερνάμε και αυξάνουμε το μετρητή j για να δείξουμε ότι πάμε στην επόμενη γραμμή στο αρχικό μητρώο. Αν η γραμμή που βρισκόμαστε δεν είναι κενή τότε το περιεχόμενο της (οι αριθμοί που περιέχονται στη λίστα- docIDs) μας δείχνουν ποιες στήλες στο μητρώο υπογραφών θα επηρεαστούν. Έπειτα, για κάθε μία μετάθεση αναλόγως τη «γραμμή»(j) που βρισκόμαστε στον πίνακα αρχικού μητρώου αναζητούμε τον αριθμό «j» στις μεταθέσεις. Επειδή ο πίνακας μεταθέσεων περιέχει K λεξικά, ψάχνουμε σε κάθε λεξικό το κλειδί «j» για να βρούμε την τιμή του(value). Αυτή η τιμή μας προσδιορίζει την θέση στην συγκεκριμένη μετάθεση . Η τιμή αυτή του λεξικού στο SIG θα μπει στη θέση [αριθμός μετάθεσης][docID],όπου «αριθμός μετάθεσης» είναι η μετάθεση που διαπραγματευόμαστε(λεξικό) και «docID» είναι κάθε φορά το πρώτο στοιχείο της λίστας row. Ταυτόχρονα, το value του λεξικού θα μπεί στο μητρώο υπογραφών εφόσον είναι μικρότερο από το προηγούμενο που μπήκε στην συγκεκριμένη θέση στο SIG.

```
105 def MyMinHash(wordset,K):
106     SIG = []
107     wordList = [[]]*int(W)              # edw 8a ftiaxoume to arxiko
108     wordLists = [[]]*int(numDocuments)  # edw ta frozensets tha gino
109     # initialize register of signatures(diplo for diafaneies)
110     rows, cols = (K, len(wordset))
111     # arxikopoihsh SIG sto apeiro gia na mporoume na kanoume allages
112     SIG = [[float('inf') for i in range(cols)] for j in range(rows)]
113     # sortaroume tis listes eswterika gia na ftiaksoume pinaka arxik
114     docID = 0
115     for fSet in wordset:                        # prospername ta fro
116         wordLists[docID] = sorted(fSet)         # sortaroume tis lis
117         for wordID in wordLists[docID]:         # bazoume ta wordID
118             if (wordList[wordID-1] == []):
119                 wordList[wordID-1] = [docID+1]  # an den uparxei hdh
120             else:
121                 wordList[wordID-1] += [docID+1] # an uparxei hdh kan
122         docID += 1
123     # ftiaxnoume mhtrwo ypografwn
124     # sarwnoume to mhtrwou eggrafwn gia na ftiaxtei mhtrwo ypografwn
125     j = 0       # gia na exoume to wordId (thesh) sto arxiko mhtrwo
126     # shmeiwsh : to pinaka me to arxiko mhtrwo ton ksekinhsame apo t
127     for row in wordList:
128         while(row != []):       # epeidh dn kseroume posa stoixeia m
129             for i in range(K):
130                 value = permutations[i][j]      # to proswrino value
131                 if(value < SIG[i][row[0]-1]):   # elegxos an h kaino
132                     SIG[i][row[0]-1] = value    # apothikeuoume th k
133             row.remove(row[0])  # diagrafoume to prwto stoixeio[apo
134         j += 1
135     return SIG
```

Παρακάτω παρατίθεται ένα παράδειγμα για το τρόπο που αναπαρίστανται οι πίνακες σχηματικά στο πρόγραμμά μας. (το παράδειγμα δεν περιέχει σωστά αποτελέσματα στο μητρώο υπογραφών) :



Βήμα 4 : Ομοιότητα Υπογραφών

Στην συνάρτηση αυτή βάζουμε ως είσοδο 2 στήλες του μητρώου υπογραφών που αντιστοιχούν στα 2 διαφορετικά docIDs που θέλουμε να ελέγξουμε την ομοιότητά τους. Επιπλέον, δίνουμε την

δυνατότητα στο χρήστη να επιλέξει πόσες γραμμές των υπογραφών των εγγράφων θέλει να συγκρίνει (1≤ numPermutations ≤K) . Αυτό έχει ως αποτέλεσμα όσο μεγαλώνει η τιμή του numPermutations η τιμή της Signature Similarity να προσεγγίζει την τιμή της Jaccard Similarity. Αν το numPermutations = K τότε Jaccard Similarity = Signature Similarity.

```
137 def MySigSim(doc1, doc2, numPermutations):
138     sigSim = 0
139     for i in range(numPermutations):
140         if(doc1[i] == doc2[i]):
141             sigSim += 1
142     sigSim = sigSim / numPermutations
143     return sigSim
```

Βήμα 5 : Υπολογισμός Πλησιέστερων Γειτόνων ανά Έγγραφο

α) Μέθοδος ωμής βίας

Εδώ δείχνουμε πως καλείται η μέθοδος της ωμής βίας από την main για τις δύο διαφορετικές μετρικές. Στη κάθε μετρική καλούμε όλα τα έγγραφα εκτός από το τελευταίο. Δεν χρειάζεται να το συγκρίνουμε με κάποιο άλλο έγγραφο καθώς έχει ήδη συγκριθεί σε προηγούμενες επαναλήψεις.

Jaccard Similarity : καλούμε την neighborsBF με ταξινομημένη λίστα για διαφορετικό κάθε φορά docID. Η λίστα περιέχει τα wordIDs του συγκεκριμένου docID

```
442     startTime = time()
443     for i in range(numDocuments-1):         #numDocuments-1 den theloume to teleutaio egrrafo
444         avgDocID = neighborsBF(numDocuments, NumNeighbors, choose, wordListSorted[i], i+1, K)
445         avgSimDocIDs += avgDocID
446     avgSimDocIDs = avgSimDocIDs / numDocuments
447     endTime = time()
448     print("Execution time : %f\n" %(endTime - startTime))
449     print("Average Similarity = ", avgSimDocIDs)
```

Signature Similarity : καλούμε την neighborsBF με λίστα για διαφορετικό κάθε φορά docID. Η λίστα signatureList είναι μια τροποποίηση της SIG(μητρώο υπογραφων) για να μπορούμε να έχουμε κάθε φορά την στήλη που θέλουμε για τον υπολογισμό της sigSim.

```
454     startTime = time()
455     for i in range(numDocuments-1):     #numDocuments-1 den theloume to teleutaio egrrafo na
456         avgDocID = neighborsBF(numDocuments, NumNeighbors, choose, signatureList[i], i+1, K)
457         avgSimDocIDs += avgDocID
458     avgSimDocIDs = avgSimDocIDs / numDocuments
459     endTime = time()
460     print("Execution time : %f\n" %(endTime - startTime))
461     print("Average Similarity = ", avgSimDocIDs)
```

Στη μέθοδο ωμής βίας αρχικά εισάγουμε ένα docID για το οποίο θα υπολογίσουμε τους γείτονές του. Ταυτόχρονα, μέσω ενός flag(choose) επιλέγουμε αν θα χρησιμοποιήσουμε την jaccard μετρική ή την signature μετρική για τον υπολογισμό της ομοιότητας. Αποθηκεύουμε σε μία λίστα(distancesList) το άλλο docID που συγκρίθηκε με το αρχικό και στην ακριβώς επόμενη θέση αποθηκεύουμε την απόστασή τους.

Στη συνέχεια, την λίστα αυτή (distancesList) την αποθηκεύουμε σε ένα λεξικό(distancesDict) που δεικτοδοτείται με κλειδιά τα αναγνωριστικά εγγράφων και τιμές τις αποστάσεις αυτών των εγγράφων από το docID. Έπειτα, ταξινομούμε το λεξικό αυτό ως προς τις τιμές απόστασης δηλαδή τα values και το ονομάζουμε dict1. Αυτό το κάνουμε για να μπορέσουμε να έχουμε στις πρώτες θέσεις του λεξικού τα docIDs που έχουν την μικρότερη απόσταση(μεγαλύτερη ομοιότητα) από το αρχικό docID.

Στη γραμμή 170-176, για το αρχικό docID, επιλέγουμε όσους neighbors ζήτησε ο χρήστης και τους προσθέτουμε σε ένα λεξικό(myNeighborsDictBF) με κλειδί το κάθε φορά γειτονικό του docID και σαν τιμή την ομοιότητά τους. Μετά, για να υπολογίσουμε ένα απλό μέτρο εγγύτητας του συγκεκριμένου εγγράφου docID με τους γειτονές του, υπολογίζουμε τον μέσο όρο ομοιότητας προσθέτοντας αρχικά τις τιμές των γειτόνων του και έπειτα διαιρώντας με το πλήθος τους. Τους μέσους όρους αυτούς,τους επιστρέφουμε στην main και τους αθροίζουμε και για να προκύψει ο τελικός μέσος όρος των μέσων εγγύτητας διαιρούμε με το συνολικό πλήθος των εγγράφων που επιλέξαμε να επεξεργαστούμε(numDocuments).

```
145 def neighborsBF(numDocuments, NumNeighbors, choose, wordlist, d, numPermutations):
146     myNeighborsDictBF = {}
147     distancesDict = {}
148     distancesList = []
149     # Sthn periptwsh pou o xrhsths epelexe thn Jaccard metrikh
150     if(choose == 1):
151         for i in range(d,numDocuments):
152             jacSimOrdered = MyJacSimWithOrderedLists(wordlist, wordListSorted[i])   # Jaccard Similari
153             distanceJac = 1 - jacSimOrdered                                         # Jaccard Distance
154             distancesList.append(i+1)                                               # bazoume ton metr
155             distancesList.append(distanceJac)                                       # etsi wste na ein
156     # Sthn periptwsh pou o xrhsths epelexe thn Signature metrikh
157     else:
158         for i in range(d,numDocuments):
159             sigSim = MySigSim(wordlist, signatureList[i], numPermutations)                    # defa
160             distanceSigSim = 1 - sigSim
161             distancesList.append(i+1)
162             distancesList.append(distanceSigSim)
163     # metaroph ths listas se dioctionary opou kleidi einai to docID twn allwn eggrafwn se sxesh me to
164     # docID pou to balame gia na broume tous geitones tou kai timh h apostash me to sygkrekrimeno doci
165     distancesDict = {distancesList[i]: distancesList[i + 1] for i in range(0, len(distancesList), 2)}
166     #sorted dictionary basei twn apostasewn
167     dict1 = dict(sorted(distancesDict.items(), key=lambda item: item[1]))
168     c = 0
169     # pairnoume tous kontinoterous geitones gia kathe kleidi(key)
170     for key, value in dict1.items():
171         if(c < NumNeighbors):
172             # metatrepoume to distance se similarity
173             myNeighborsDictBF[key] = 1 - value
174             c += 1
175         else:
176             break
177     avgDocID = 0
178     # prosthetoume ola ta similarities twn kontinoterwn geitonwn
179     for value in myNeighborsDictBF.values():
180         avgDocID += value
181     avgDocID = avgDocID / len(myNeighborsDictBF)
182     if (choose == -1):  # xrhsimopoieitai gia na broume toys geitones enow sugkekrimenou docID
183         return myNeighborsDictBF
184     return  avgDocID
```

β)Μέθοδος LSH

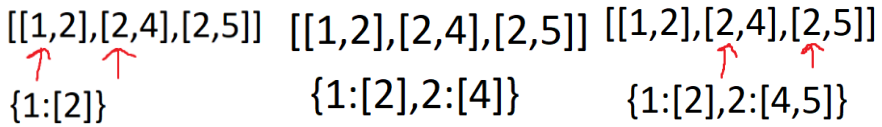
Πρώτη σημείωση για τον LSH είναι πως δεν χρησιμοποιούμε αυτούσιο τη SIG. Χρησιμοποιούμε την signatureList η οποία έχει τα ίδια περιεχόμενα με τη SIG, η οποία έχει σαν αναπαράσταση πιο εύχρηστα τις στήλες.

Αρχικά, δημιουργούμε μόνο μία τυχαία συνάρτηση κατακερματισμού η οποία θα χρησιμοποιηθεί για όλες τις μπάντες, αυτό το κάνουμε ώστε σε διαφορετικές μπάντες οι ίδιες πλειάδες να μπαίνουν στον ίδιο κάδο. Έπειτα επεξεργαζόμαστε κάθε μπάντα. Ξεκινώντας, για κάθε μπάντα έχουμε μια άδεια λίστα(listLSH) και έναν μετρητή(findDocID) που μας δείχνει κάθε φορά σε ποιο έγγραφο(στήλη) στο μητρώο υπογραφών βρισκόμαστε, για αυτό για κάθε μπάντα αρχικοποιείται στο 1. Στη συνέχεια, διατρέχουμε στήλη-στήλη το μητρώο υπογραφών και για κάθε στήλη της μπάντας παίρνουμε τις γραμμές της(rowsPerBands). Τις γραμμές αυτές τις κάνουμε πλειάδα και τις περνάμε ως είσοδο στην συνάρτηση hash(γραμμή 215) για να μας δώσει μία τιμή. Αυτή την τιμή την δίνουμε στην τυχαία συνάρτηση κατακερματισμού που έχουμε φτιάξει (hashLSH) και μας υποδεικνύει σε ποιο κάδο ανήκει το συγκεκριμένο docID στην συγκεκριμένη μπάντα. Στη συνέχεια, αποθηκεύουμε σε ένα λεξικό(dictLSH), το οποίο έχει για κλειδί το docID και τιμή το κάδο αυτόν. Το λεξικό αυτό το ταξινομούμε με βάση το κάδο και με αυτό το τρόπο επιτυγχάνουμε κλειδιά(docIDs) που έχουν πέσει στον ίδιο κάδο να βρίσκονται δίπλα δίπλα(όνομα λεξικού newDict). Έπειτα το λεξικό αυτό το μετατρέπουμε σε μία λίστα (listLSH) ώστε να είναι πιο εύκολο να εντοπίσουμε τα υποψήφια ζεύγη(με μία for). Στη λίστα αυτή περνάμε με την σειρά docID(κλειδί) και κάδο(value). Στο τέλος της επεξεργασίας κάθε μπάντας, ελέγχουμε κατά ζεύγη τους κάδους της λίστας(listLSH) διότι γνωρίζουμε πως βρίσκονται σε διπλανές θέσεις στη λίστα και προσθέτουμε σε μια νέα λίστα-λιστών(candidatePairs) τα ζεύγος των docIDs που είχαν την ίδια τιμή κάδου. Ακόμα ελέγχουμε πως το κάθε ζεύγος θα αποθηκεύεται μόνο μία φορά στο candidatePairs.

Σημειώνουμε : στα candidatePairs για κάθε ζεύγος το ‘αριστερό’ docID είναι πάντα μικρότερο του ‘δεξιού’. Αυτό συμβαίνει λόγο του ότι είχαμε ταξινομήσει το λεξικό newDict παραπάνω.

```
198 def LSH(signatureList, rowsPerBands,choose):
199     #kleidi -> apo 1 mexri numDoc (den 8a xrhsimopoihsw to SIG voleuei sthles)
200     #value  -> timh pou antistoixei ton kado pou mpainei to kathe docID
201     global myNeighborsDict
202     dictLSH = {}
203     candidatePairs = []
204     numBands = math.floor(K/rowsPerBands)       # an prolavainoume na koitaksoume thn teleutaia mpanta
205     counter = 0
206     findDocID = 1
207 
208     hashLSH = create_random_hash_function()
209     for i in range(numBands):
210         listLSH = []                        # adeiazoume kathe fora th lista wste na apothikeutei to l
211         findDocID = 1                       # na mpainei kathe fora to swsto docID gia key
212         for col in signatureList:
213             vector = col[0+counter:rowsPerBands+counter]    # pairnoume rowsPerBands grammes ths sth1h
214             vector_tuple = tuple(vector)                    # tis metatrepoume pleiada
215             hashValue = hash(vector_tuple)                  # exoume thn timh
216             tmp = hashLSH(hashValue)                        # o kados
217             dictLSH[findDocID] = tmp                        # pairnoume sto leksiko
218             findDocID += 1                                  # auksanoume to docd id gia na pame sthn e
219         # sortaroume wste ta upopshfia zeugh na einai o enas meta ton allon
220         newDict = {k: v for k, v in sorted(dictLSH.items(), key=lambda item: item[1])}
221         [listLSH.extend([k,v]) for k,v in newDict.items()]  # kanoume lista to sortarismeno dict
222         counter += rowsPerBands                             # na proxwraei kata rowsPerBands kathe for
223         for i in range(1,len(listLSH)-2,2):                 # gia kathe mpanta briskoume ta upospshfia
224             # elegxoume an einai upopshfio zeugos kai tote apothikeuoume to docID tous
225             # elegxoume an uparxei hdh to zeugos mesa sto candidate pairs
226             if((listLSH[i] == listLSH[i+2]) and ([listLSH[i-1], listLSH[i+1]] not in candidatePairs)):
227                 candidatePairs.append([listLSH[i-1], listLSH[i+1]])
228     candidatePairs = sorted(candidatePairs)
```

Στη γραμμή 231 ταξινομούμε τη λίστα λιστών(candidatePairs). Με αυτό το τρόπο έχουμε ταξινομημένα όλα τα ζευγάρια που μας βοηθάει ώστε να γνωρίζουμε ότι οι γείτονες των ζευγών θα βρίσκονται ο ένας μετά τον άλλον. Συνεχίζοντας, διατρέχουμε τη λίστα-λιστών και αν το πρώτο στοιχείο(‘αριστερο’) από κάθε ζεύγος δεν είναι ίδιο με το προηγούμενο πρώτο στοιχείο(‘αριστερο’) τότε το ζεύγος που ελέγχουμε το βάζουμε σε ένα λεξικό(myNeighborsDict), διότι το πρώτο του στοιχείο αποτελεί νέο κλειδί για το λεξικό μας. Αν όμως το στοιχείο αυτό (‘αριστερό στοιχείο’) είναι ίδιο με το προηγούμενο (‘αριστερό στοιχείο’) τότε προσθέτουμε στο λεξικό για το κλειδί(‘αριστερό στοιχείο’) το ‘δεξιό του στοιχείο’.




```
228     candidatePairs = sorted(candidatePairs)
229     # apo thn lista listwn pername se leksiko kleidi me times
230     for i in range(0,len(candidatePairs)):
231         # elegxoume an h prwth timh einai idia me prohgoumenh timh (dhladh uparxei h
232         if (candidatePairs[i-1][0] == candidatePairs[i][0]):
233                 myNeighborsDict[candidatePairs[i][0]].append(candidatePairs[i][1])
234         else:
235             # einai gia opoiodhpote zeugos den exei ksanampei mesa sto dict (apothik
236             myNeighborsDict.setdefault(candidatePairs[i][0], [candidatePairs[i][1]])
237     AvgSimDocID = 0
238     for key,value in myNeighborsDict.items():
239         avgDocID = neighborsLSH(NumNeighbors, choose, key, value, K)
240         AvgSimDocID += avgDocID
241     AvgSimDocID = AvgSimDocID / numDocuments
242     return AvgSimDocID
```

Τέλος ακολουθούμε τα βήματα όπως περιεγράφηκαν στην ωμή βία(neighborsBF), η διαφορά είναι πως τώρα δεν θα υπολογίσουμε ομοιότητες για όλα τα άλλα docIDs, παρά μόνο για αυτά που έχουν περάσει ένα συγκεκριμένο threshold.

```
244 def neighborsLSH(NumNeighbors, choose, keyDict, valuesDict, numPermutations):
245     distancesDict = {}
246     distancesList = []
247     myNeighborsDict1 = {}
248     if (choose == 1):
249         for i in valuesDict:
250             jacSimOrdered = MyJacSimWithOrderedLists(wordListSorted[keyDict-1], wordListSorted[i-1])
251             distanceJac = 1 - jacSimOrdered
252             distancesList.append(i+1)               #epeidh oi listes xekinane apo to 0 kai emeis 8el
253             distancesList.append(distanceJac)
254     else:
255         for i in valuesDict:
256             sigSim = MySigSim(signatureList[keyDict-1], signatureList[i-1], numPermutations)
257             distanceSigSim = 1 - sigSim
258             distancesList.append(i+1)               #epeidh oi listes xekinane apo to 0 kai emeis 8el
259             distancesList.append(distanceSigSim)
260     distancesDict = {distancesList[i]: distancesList[i + 1] for i in range(0, len(distancesList), 2)}
261     dict1 = dict(sorted(distancesDict.items(), key=lambda item: item[1]))
262     c = 0
263     for key, value in dict1.items():
264         if(c < NumNeighbors):
265             myNeighborsDict1[key] = 1 - value
266             c += 1
267         else:
268             break
269     avgDocID = 0
270     for value in myNeighborsDict1.values():
271         avgDocID += value
272     avgDocID = avgDocID / len(myNeighborsDict1)
273     return avgDocID
```

Σημείωση : δεν έχουμε προλάβει να ενσωματώσουμε στο κώδικά μας για τον αριθμό rowsPerBands πως για την τελευταία μπάντα να παίρνει το πολύ 2r-1 γραμμές. Ο κώδικας έχει βρεθεί και παρατίθεται παρακάτω.

```
if (K % rowsPerBands != 0):
    # to teleutaio numDocument theloume K + (K%rowsPerBands)
numdocuments = math.floor(K/rowsPerBands)
```

Παρατηρήσεις:

- Η LSH μέθοδος είναι αισθητά πιο γρήγορη από την brute force λόγω του ότι υπολογίζουμε ομοιότητες μόνο για τα έγγραφα που έχουν περάσει ένα συγκεκριμένο threshold και όχι όλα τα έγγραφα που πολλά από αυτά είναι λογικό να είναι εντελώς ανόμοια. Άρα με αυτό τον τρόπο γλιτώνουμε υπολογισμούς ομοιότητας οι οποίοι στο τέλος δεν θα τους χρησιμοποιήσουμε επειδή δεν θα είναι κοντινοί γείτονες για τα έγγραφα που συγκρίναμε.
- Δεν καταφέραμε να παραμετροποιήσουμε τα rows per Band για συγκεκριμένη τιμή του K(πλήθος μεταθέσεων) όπως υποδείξατε στην απάντηση της ερώτησης 6. Παρόλα αυτά , παρατηρήσαμε ότι για σταθερό K και μεταβάλλοντας το r(rowsPerBand) σε μικρότερη τιμή μειώνεται το s(threshold) και αντίθετα . Όπως, και κρατώντας σταθερό το r και μειώνοντας το K τότε το s αυξάνεται και αντίθετα.
- Ακόμα παρατηρήσαμε πως η Signature Similarity είναι πιο αποδοτική σε θέμα χρόνου έναντι της Jaccard Similarity αυτό συμβαίνει γιατί παίρνει ως είσοδο μια ‘υπογραφή’ η οποία είναι μικρότερο σύνολο του αρχικού μητρώου. Όμως η Jaccard Similarity λόγω αυτού είναι πιο ακριβής σαν μέθοδος.

Επιπλέον, για την επιλογή που έχει ο χρήστης ώστε να αφήνει το πρόγραμμα να βρίσκει την καλύτερη παραμετροποίηση για καλύτερο αποτέλεσμα στην LSH βάλαμε μια default τιμή και συγκεκριμένα την τιμή 2.

- Σημαντικό! Κάθε φορά που θέλουμε να αλλάξουμε το πλήθος των μεταθέσεων σε σχέση με την προηγούμενη εκτέλεση του προγράμματος θα πρέπει να διαλέγουμε από το μενού επιλογής το «2» έτσι ώστε να κάνουμε update το ήδη υπάρχων έγγραφο που έχουμε δημιουργήσει για τις μεταθέσεις. Επίσης το ίδιο ισχύει ακόμα και αν θέλουμε να διαβάσουμε από διαφορετικό αρχείο Bag-of-Words σε σχέση με την προηγούμενη εκτέλεση του κώδικα. Είτε μπορούμε να διαγράφουμε το αρχείο χειροκίνητα και να πατήσουμε στο μενού επιλογής το «1».

Για αυτό σημειώνουμε πως οι τιμές που βγάλαμε στο Experiment το hashFunctions.csv είναι για το Enron ενώ το hashFunctions(1).csv είναι για το Nips